Data intensive ATLAS workflows in the Cloud

Dissertation

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
„Doctor rerum naturalium“
der Georg-August-Universität Göttingen

im Promotionsprogramm ProPhys
der Georg-August University School of Science (GAUSS)

vorgelegt von

Gerhard Ferdinand Rzehorz

aus Bruchsal

Göttingen, 2018

Betreuungsausschuss

Prof. Dr. Arnulf Quadt
PD. Dr. Jörn Große-Knetter

Mitglieder der Prüfungskommission:

Referent:	Prof. Dr. Arnulf Quadt II. Physikalisches Institut, Georg-August-Universität Göttingen
Koreferent:	Prof. Dr. Ramin Yahyapour Institut für Informatik, Georg-August-Universität Göttingen

Weitere Mitglieder der Prüfungskommission:

Prof. Dr. Steffen Schumann
II. Physikalisches Institut, Georg-August-Universität Göttingen

Prof. Dr. Jens Grabowski
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Ariane Frey
II. Physikalisches Institut, Georg-August-Universität Göttingen

Dr. Oliver Keeble
IT Department, CERN

Tag der mündlichen Prüfung: 09.05.2018

Data intensive ATLAS workflows in the Cloud

Abstract

Large physics experiments, such as ATLAS, have participating physicists and institutes all over the Globe. Nowadays, physics analyses are performed on data that is stored thousands of kilometres away. This is possible due to the distributed computing infrastructure known as the Worldwide LHC Computing Grid (WLCG). In addition to the analyses, all the previous data transformation steps, such as raw data reconstruction, are performed within the WLCG. Within the next decade, the computing requirements are projected to exceed the available resources by a factor of ten. In order to mitigate this discrepancy, alternative computing solutions have to be investigated. Within this thesis, the viability of Cloud computing is evaluated. The concept of Cloud computing is to rent infrastructure from a commercial provider. In contrast to that, in the WLCG computing concept the hardware within the computing centres is purchased and operated by the WLCG. In order to examine Cloud computing, a model that predicts the workflow performance on a given infrastructure is created, validated and applied. In parallel, the model was used to evaluate a workflow optimisation technique called overcommitting. Overcommitting means that the workload on a computer consists of more parallel processes than there are CPU cores. This technique is used to fill otherwise idle CPU cycles and thereby increase the CPU utilisation. Using the model, overcommitting is determined to be a viable optimisation technique, especially when using remote data input, taking into account the increased memory footprint. Introducing the overcommitting considerations to the Cloud viability evaluation increases the feasibility of Cloud computing. This is because Cloud computing may not include a storage solution and has the flexibility to provision virtual machines with additional memory. The final conclusion is drawn by taking the above described results and by combining them with the cost of the WLCG and the Cloud. The result is that Cloud computing is not yet competitive compared to the WLCG computing concept.

Data intensive ATLAS workflows in the Cloud

Zusammenfassung

Die großen Physikexperimente, wie zum Beispiel ATLAS, bestehen aus Kollaborationen mit Physikern und Instituten auf der ganzen Welt. Heutzutage werden physikalische Analysen an Daten durchgeführt, die Tausende von Kilometern entfernt gespeichert sind. Dies ist aufgrund der verteilten Computing-Infrastruktur, die als Worldwide LHC Computing Grid (WLCG) bekannt ist, möglich. Zusätzlich zu den Analysen werden alle vorherigen Datentransformationsschritte, wie die Rekonstruktion von Rohdaten, innerhalb des WLCG durchgeführt. Innerhalb des nächsten Jahrzehnts wird erwartet, dass die Anforderungen an die Computerinfrastruktur die verfügbaren Ressourcen um den Faktor zehn übersteigen werden. Um diese Diskrepanz zu mindern, müssen Alternativen zur jetzigen Computerinfrastruktur untersucht werden. Im Rahmen dieser Arbeit wird Cloud Computing evaluiert. Das Konzept von Cloud Computing besteht darin, eine Computerinfrastruktur von einem kommerziellen Anbieter zu mieten. Dies steht im Gegensatz zum WLCG Konzept, in dem die Ausstattung der Rechenzentren gekauft und selbst betrieben wird. Um Cloud Computing zu untersuchen, wird ein Modell erstellt, validiert und angewendet, dass das Verhalten von Arbeitsflüssen auf einer beliebigen Infrastruktur vorhersagt. Parallel dazu wurde das Modell zur Bewertung einer Arbeitsfluss-Optimierungsmethode namens Overcommitting verwendet. Overcomitting bedeutet, dass die Arbeitslast auf einem Computer aus mehr parallelen Prozessen besteht, als CPU-Kerne vorhanden sind. Diese Technik wird verwendet, um ansonsten ungenutzte CPU-Zyklen zu füllen und dadurch die CPU-Auslastung zu erhöhen. Unter der Verwendung des Modells wird das Overcommitting als eine brauchbare Optimierungstechnik ermittelt. Dies gilt insbesondere dann, wenn die Daten nur auf weit entfernten Speichermedien vorhanden sind und unter der Berücksichtigung des erhöhten Bedarfs an Arbeitsspeicher. Der Einbezug dieser Überlegungen in die Cloud Computing Evaluation verbessert dessen Stellung. Dies liegt daran, dass Cloud Computing nicht unbedingt Speichermöglichkeiten enthält und flexibel genug ist, um virtuellen Maschinen zusätzlichen Arbeitsspeicher zuzuweisen. Unter Berücksichtigung all dieser Gesichtspunkte und in Kombination mit den Kostenmodellen des WLCG und der Cloud, ergibt sich, dass Cloud Computing noch nicht konkurrenzfähig gegenüber dem bisherigen WLCG Konzept ist.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Thesis structure . . . 2

2 The Standard Model of particle physics **3**
- 2.1 Interactions . . . 3
  - 2.1.1 Weak interaction . . . 3
  - 2.1.2 Electromagnetic interaction . . . 5
  - 2.1.3 Electroweak unification . . . 5
  - 2.1.4 Strong interaction . . . 5
  - 2.1.5 Quarks and leptons . . . 6
  - 2.1.6 The Higgs mechanism . . . 6
- 2.2 Beyond the Standard Model . . . 7

3 The ATLAS detector **9**
- 3.1 LHC . . . 9
  - 3.1.1 CERN . . . 9
  - 3.1.2 Machine specifics . . . 9
- 3.2 ATLAS . . . 12
  - 3.2.1 Detector components . . . 12
  - 3.2.2 Inner detector . . . 13
  - 3.2.3 Calorimeters . . . 13
  - 3.2.4 Muon spectrometer . . . 14
  - 3.2.5 Trigger and data acquisition . . . 15

4 LHC offline computing **17**
- 4.1 Distributed and Grid computing . . . 18
- 4.2 Cloud computing . . . 19
  - 4.2.1 Concept . . . 20

4.2.2 Pricing . . . 22
4.2.3 Storage . . . 24
4.2.4 Security, safety and integrity . . . 27
4.2.5 Availability . . . 28
4.3 Grid Computing . . . 29
4.4 WLCG . . . 29
4.4.1 Concept and purpose . . . 30
4.4.2 Composition . . . 31
4.4.3 Evolution . . . 37
4.5 ATLAS computing components . . . 37
4.5.1 XRootD . . . 37
4.5.2 Athena . . . 38
4.5.3 AthenaMP . . . 38
4.5.4 PanDA . . . 38
4.5.5 Rucio . . . 39
4.5.6 JEDI . . . 40
4.5.7 CVMFS . . . 40
4.5.8 Tags . . . 40
4.5.9 AMI . . . 40
4.6 General concepts . . . 40
4.6.1 Benchmarking . . . 40
4.6.2 Storage . . . 41
4.6.3 Swapping . . . 42
4.6.4 CPU efficiency . . . 43
4.6.5 Undercommitting . . . 43
4.6.6 Control groups . . . 44

5 Workflows **45**
5.1 General model . . . 47
5.1.1 All experiments . . . 49
5.1.2 ATLAS . . . 50
5.2 Monte Carlo simulation . . . 51
5.2.1 Event generation . . . 51
5.2.2 Simulation . . . 54
5.3 Reconstruction . . . 56
5.3.1 Raw data reconstruction . . . 57
5.3.2 Raw data reconstruction profile . . . 57
5.3.3 Simulated data reconstruction . . . 64
5.3.4 Digitisation . . . 66
5.3.5 Trigger simulation . . . 67
5.3.6 Reprocessing . . . 67
5.4 Analysis . . . 68
5.4.1 Group production . . . 69

5.4.2 Complete processing . . . 69

6 Models and predictions **71**
6.1 Related work . . . 72
6.2 The Workflow and Infrastructure Model . . . 73
6.2.1 Functionalities . . . 74
6.2.2 Model input . . . 76
6.3 Model logic . . . 78
6.3.1 Workflow duration . . . 78
6.3.2 CPU consumption time . . . 79
6.3.3 Idle time . . . 80
6.3.4 I/O wait time . . . 80
6.3.5 Overhead time . . . 81
6.3.6 Swap time . . . 82
6.3.7 Undercommitted idle time . . . 82
6.3.8 Number of machines . . . 83
6.3.9 Final result . . . 83
6.3.10 Estimation of uncertainties . . . 84
6.4 Programming tools . . . 85
6.5 Complex workflow model . . . 85

7 Cloud workflow modelling **87**
7.1 Validation . . . 87
7.1.1 Strategy . . . 87
7.1.2 Setup . . . 90
7.1.3 Validation - workflow fluctuation . . . 92
7.1.4 Results . . . 94
7.1.5 Conclusion . . . 97
7.2 Cloud measurement . . . 97
7.2.1 HNSciCloud: large scale . . . 98
7.2.2 HNSciCloud: object storage . . . 104
7.2.3 Grid sites . . . 107
7.3 Model application . . . 108
7.3.1 Combining benchmarks . . . 109
7.3.2 Bandwidth estimation . . . 110
7.3.3 HNSciCloud large scale . . . 111
7.3.4 Error sources . . . 111

8 Overcommitting **117**
8.1 Principle . . . 117
8.1.1 Overcommitting scenarios . . . 119
8.2 Study . . . 122
8.2.1 Overcommitting with AthenaMP . . . 122
8.2.2 Overcommitting job profiles . . . 123

8.3 Measurements . . . 126
8.3.1 AthenaMP combinations . . . 126
8.3.2 Overcommitting for latency hiding . . . 130
8.3.3 Scheduling . . . 131
8.4 Model predictions . . . 132
8.4.1 Overcommitting in the model . . . 132
8.4.2 Result . . . 133
8.5 Conclusion . . . 135

9 Cloud viability evaluation **137**
9.0.1 Model results . . . 139
9.1 Conclusion . . . 140

10 Summary and Outlook **141**

Acknowledgements **143**

Bibliography **145**

Appendices **155**
A.1 Job specifications . . . 157
A.1.1 Workflows: Event generation . . . 157
A.1.2 Workflows: Raw data reconstruction . . . 162
A.2 Additional profiles . . . 163
A.3 Model implementation . . . 167
A.3.1 Model usage . . . 168
A.3.2 Model code . . . 169
A.4 Overcommitting . . . 192
A.4.1 Model input parameters . . . 193
A.5 Additional source code and scripts . . . 193
A.6 Different workflows . . . 195
A.6.1 Event Generation . . . 195
A.6.2 Monte-Carlo simulation . . . 196
A.6.3 Reconstruction 1 . . . 196
A.6.4 Reconstruction 2 . . . 197
A.6.5 Reconstruction 3 . . . 198
A.6.6 Reconstruction 4 . . . 199
A.6.7 Reconstruction 5 . . . 200
A.6.8 Reconstruction 6 . . . 201
A.6.9 Reconstruction 7 . . . 204
A.6.10 Digitisation and reconstruction 1 . . . 205
A.6.11 Digitisation and reconstruction 2 . . . 207
A.6.12 Digitisation and reconstruction 3 . . . 209
A.6.13 Digitisation and reconstruction 4 . . . 212

A.6.14 Digitisation and reconstruction 5 215
A.7 Hardware 215
A.7.1 Göttingen 215
A.7.2 CERN 216
A.7.3 Exoscale 217
A.7.4 IBM 218
A.7.5 T-systems 219
A.8 Additional Tables 220

CHAPTER 1

Introduction

It is in our human nature to be curious and to thirst for knowledge. These human characteristics combined with a complex world result in the field of physics. The ultimate goal of physics is to have a complete and accurate description of the universe and everything in it. In order to make such a description, many experiments and measurements are necessary. Nowadays, the boundary of the unknown has been pushed to include particles that are infinitely small, according to our current understanding, but is far from complete.

The search for new particles in the 21st century is taking place through cosmic observations and terrestrial particle acceleration. With recent advancements in technology, the scale of these experiments has increased massively. Not only are the experimental setups several kilometres in size, but the amounts of data that are being collected is also enormous.

These efforts culminated in the discovery of the Higgs Boson. The Higgs Boson was one of the missing pieces, that were needed in order to complete the Standard Model and the understanding of the universe. Finding the Higgs Boson was, however, only one of the many purposes of the ongoing High Energy Physics (HEP) experiments. There are still a multitude of open questions about the universe, which to this day, remain unanswered.

1.1 Motivation

The times in which a computer could handle the data processing of an experiment have long since passed. Nowadays, thousands of interconnected Central Processing Units (CPUs) are necessary to keep up with the ratio at which physics data is collected. There is no end in sight, to the steady increase in data that is analysed to further the understanding of the universe.

The increase in computational requirements can even be observed within the same experiments. According to the latest prognoses, the computing and storage requirements that the four major HEP experiments will pose, is ten times higher than what can be delivered, assuming a flat budget. There are several approaches to solve this issue. In this thesis, a cost reduction by outsourcing the computing to commercial providers is investigated.

1.2 Thesis structure

In Chapter 2 the Standard Model is introduced. It is the basis of modern particle physics. It is being experimentally tested in large experiments, one of the biggest being ATLAS, which is described in Chapter 3. The ATLAS collaboration, that consists of many physicists located around the Globe, is able to work together due to the distributed computing infrastructure. The infrastructure is explained in detail in Chapter 4, that introduces the most important components. The workflows that are enabled by this infrastructure are described in Chapter 5. They are the basis and provide the boundary conditions of a possible outsourcing of the computing infrastructure into the Cloud. In order to understand, whether this would be beneficial, and in order to optimise the whole computing infrastructure, the Workflow and Infrastructure Model was created. It is introduced in Chapter 6, where the underlying logic is described in detail. Measurements of the workflow performance within the Cloud are undertaken in Chapter 7. The same Chapter also uses the measured data to verify the model as well as applying it to different use cases. In Chapter 8, Overcommitting, an optimisation technique is investigated in detail. This is done by comparing the results of measurements and by applying the model. In Chapter 9 a final conclusion of the previous measurements and the viability of Cloud computing is drawn. At the end, a summary of all activities described within this thesis is given, see Chapter 10. The thesis concludes with an indication of the future direction of the ATLAS computing.

CHAPTER 2

The Standard Model of particle physics

The Standard Model (SM) describes the current knowledge of elementary particles and the interactions between them. A schematic overview of the SM particles is given in Figure 2.1.

There are six different types of quarks and leptons. Each have corresponding antiparticles, which are not included in the diagram. These antiparticles are similar to their corresponding particle, except that they have the opposite electric charge. The exception are neutrinos which do not carry an electric charge. Quarks and leptons are spin $\frac{1}{2}$ particles, so called fermions.

In addition, six different spin 1 particles (bosons) are included in the SM. The five force carrying bosons and their interaction are described in the following.

2.1 Interactions

Apart from gravitation, which is not included in the SM, there are three fundamental interactions. Particles can interact via the strong, the weak and the electromagnetic force. The forces can be described by fermions interacting with each other by exchanging gauge bosons, the force carriers.

These three interactions all conserve the energy, the momentum, the angular momentum, the electric charge, the weak isospin, the colour, the baryon number, and the lepton numbers.

2.1.1 Weak interaction

The weak force couples to the weak isospin. Charged as well as neutral leptons interact via the weak force which couples to the weak isospin. The force carriers of the weak

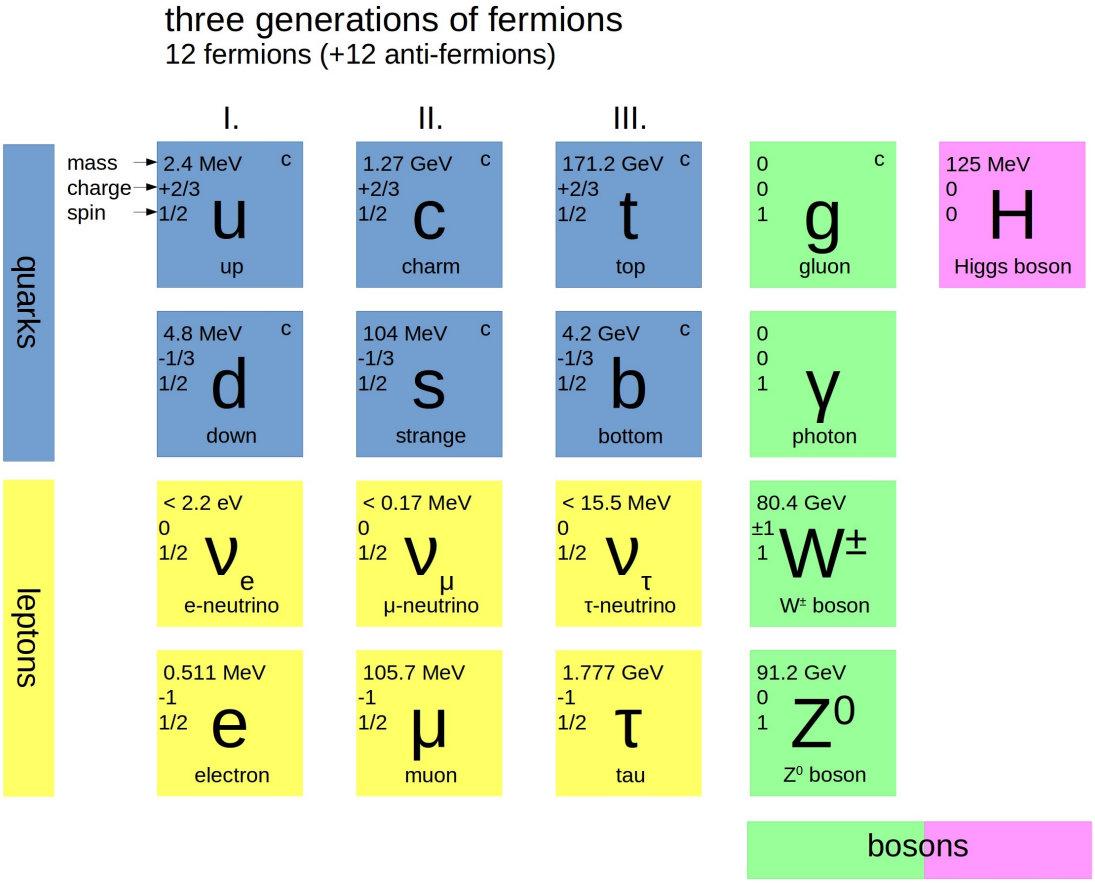

Figure 2.1: The elementary particles of the Standard Model of Particle Physics, including categorisations in quarks and leptons (in three generations) and bosons. The letter c in the upper right corner indicates particles carrying a colour charge. The indicated masses represent results from measurements.

interaction are the $W^{\pm}$ and Z^0 bosons. The high mass of these gauge bosons (80.385 ± 0.015 GeV for the $W^{\pm}$ and 91.1876 ± 0.0021 GeV for the Z^0 bosons [1]) leads to a short lifetime, which explains the short range of the weak force. The weak interaction violates the parity- (P) and the charge-parity- (CP) symmetry.

One example of a weak interaction is the β^- decay, the Feynman diagram of which can be seen in Figure 2.2. The coupling of a W^- boson to quarks as well as leptons is shown. The down quark is converted into an up quark by emitting a W^- boson which decays into electron and electron-neutrino.

The weak interaction couples only to left handed fermions and right handed antifermions. This is explained by the V-A-theory which introduces a Vector minus Axial vector Lagrangian for the weak interaction.

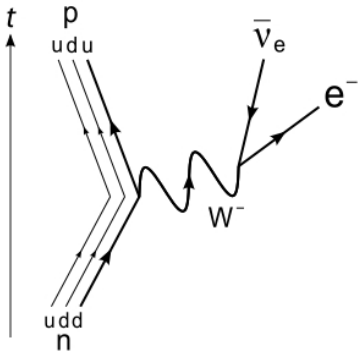

Figure 2.2: Feynman diagram of the beta minus decay, transforming a neutron into a proton (down into up quark) via the weak interaction. This process is common in unstable atomic nuclei (excess of neutrons) or free neutrons. The released electron can be identified as beta-radiation.

2.1.2 Electromagnetic interaction

The electromagnetic interaction is mediated by a massless photon and couples to the electric charge. The range of this interaction is infinite, which makes the electromagnetic force (apart from gravity) the predominantly observed force in the macroscopic and visible universe.

2.1.3 Electroweak unification

At high energies, above the electroweak unification energy of around 246 GeV, weak and electromagnetic interactions appear as one interaction. The electroweak unification theory by Glashow Salam and Weinberg [2] [3] [4], describes how these two interactions are manifestations of the same force.

According to this theory, the gauge bosons would have to be massless, which is in contradiction to the heavy mass of the $W^{\pm}$ and Z^0 bosons. This conflict is solved by the Higgs mechanism described in Subsection 2.1.6.

2.1.4 Strong interaction

The strong force is mediated by massless gluons and described by quantum chromodynamics (QCD). The gluons couple to the colour charge, which exists in three different versions (red, green, and blue). Each gluon carries a colour and an anticolour. This leads to eight different gluons, according to the combinations that are possible with three different colours and anti-colours.

The range of the strong interaction is short due to the self-interactions of the gluons. At very short distances or high energies, these self-interactions weaken the strong force (asymptotic freedom).

At long distances or low energies, the interaction becomes very strong leading to a so called confinement of the quarks.

2.1.5 Quarks and leptons

In the SM, quarks and leptons are grouped into three generations, with the same properties apart from the mass.

The quark doublet of each generation consists of an "up-like" quark (up-, charm-, top-quark) with an electric charge of $+2/3$ and a "down-like" quark (down-, strange-, bottom-quark) with an electric charge of $-1/3$. In addition to the electric charge, quarks carry also colour charge and weak isospin. Hence, they interact via the electromagnetic, weak and strong interactions. They can change their flavour via the charged weak interaction. The probability of these decays is described by the unitary Cabibbo-Kobayashi-Maskawa matrix (CKM matrix). Flavour changing neutral currents are suppressed by the Glashow-Iliopoulos-Maiani mechanism. They have been observed for the first time at the Collider Detector at Fermilab (CDF) [5].

The electrically charged leptons of the three generations are called electron, muon and tau. Since they also carry weak isospin, they interact via the electromagnetic and the weak interactions. To each of the charged leptons exists an electrically neutral neutrino. The masses of the neutrinos are found to be extremely small. They carry only weak isospin and thus interact via the weak force, which makes them difficult to detect. An explanation for the small mass could be that neutrinos are their own antiparticles (Majorana particle), which is possible because they have no electric charge.

2.1.6 The Higgs mechanism

The problem of the unexplained high mass of the $W^{\pm}$ and Z^0 bosons, which break the gauge symmetry, can be solved with the Higgs mechanism. The Higgs mechanism gives mass to the gauge bosons of the weak interaction, without introducing mass terms that are not consistent with the local gauge invariance.

It can be explained by a Higgs field, that is present everywhere. The Lagrangian for this field is:

$$\mathcal{L}_{Higgs} = (\partial_\mu \Phi)^\dagger (\partial_\mu \Phi) - V(\Phi) \tag{2.1}$$

The Higgs potential can be described by:

$$V(\Phi) = \mu^2 \Phi^\dagger \Phi + \lambda (\Phi^\dagger \Phi)^2 \tag{2.2}$$

where λ has to be positive and μ is not constrained. For $\mu^2 > 0$, the potential has only one minimum at zero, preserving the symmetry. For $\mu^2 < 0$ an infinite number of minima is prevalent and the choice of the physical vacuum expectation value spontaneously breaks the symmetry. The asymmetry of the vacuum ground state can be illustrated by looking at the Higgs potential, which has the shape of a Mexican hat, illustrated in Figure 2.3.

The spontaneous breaking of the symmetry caused by this asymmetry gives mass to the $W^{\pm}$ and Z^0 bosons. Because the photon is required to remain massless, the only free parameter left is the Higgs field which can be identified with the Higgs boson.

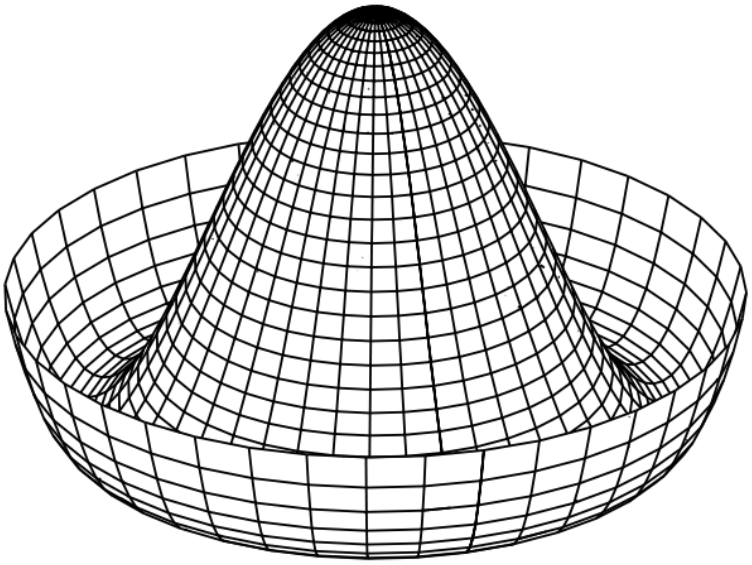

Figure 2.3: Graphical representation of the Higgs potential, where $\mu^2 < 0$.

The Higgs boson has been discovered by the ATLAS [6] and CMS [7] collaborations at the LHC [8] [9]. It has a mass of $125 \pm 0.21(stat) \pm 0.11(syst)$ GeV [10].

2.2 Beyond the Standard Model

There are phenomena that are unexplained by the SM, a few of them are stated briefly in the following.

Over one quarter of the visible universe is hypothesised to be made of dark matter. Dark matter has not been observed directly, hence the name. Strong indicators of its existence are the measurements of the cosmic microwave background and the rotational speed of galaxies, that according to calculations would need to have much more mass than is visible. There is no SM candidate for dark matter which could account for all the dark matter in the universe.

Similarly, there is no SM candidate for dark energy. Dark energy explains the observed acceleration of the universe's expansion. These observations lead to the conclusion that over $\frac{2}{3}$ of the universe consist of dark energy. It is hypothesised to exist throughout all of space, but has never been measured.

Moreover, the matter-antimatter asymmetry cannot be explained by the SM. The big bang should have created equal amounts of matter and antimatter, yet more matter than antimatter is observed. One explanation would be the charge parity (CP) violation in

weak interactions. The measured CP violations are however not big enough to explain the extent of the existing asymmetry.

Furthermore, gravity could not yet be included in the SM, but plays an important role at very high energies.

CHAPTER 3

The ATLAS detector

3.1 LHC

3.1.1 CERN

CERN, the European Organization for Nuclear Research, is home to multiple linear[1] as well as circular[2] particle accelerators. The Large Hadron Collider (LHC) is the biggest amongst them. CERN was founded in 1954 and is located close to Geneva in Switzerland, spanning across the border into France. Amongst its major achievements is the discovery of the W and Z bosons, as well as the founding of the World Wide Web. CERN is funded by contributions from its 22 member states. There are around 2500 staff members employed by CERN, only 3% of which are actually research physicists. The staff is mostly constructing, maintaining and running the machines and experiments. The ratio of physicists is much higher when looking at the over 13000 associated CERN members, that come mostly from international collaborations. They do not have to be based at CERN. The distributed computing infrastructure, see Chapter 4, enables physics analysis to be performed remotely, from anywhere in the world.

3.1.2 Machine specifics

New accelerators in particle physics are pushing the energy levels higher, in order to detect heavier unknown particles and processes. There are several possibilities to reach these energies, the first one is to achieve a better acceleration. Acceleration is achieved through electric fields, which only work on electrically charged particles. The more energy a particle should have, the longer it has to be accelerated.

[1] E.g.: LINAC3
[2] E.g. PS, SPS

The size and therefore the material costs are a limit when considering linear colliders, where the length determines the energy.

In a circular collider, the acceleration length is increased, by having the particles go around the same ring multiple times. This is limited by the circular track on which the particles have to be kept. With increasing energy, either the magnets keeping the particles on the circular orbit through the Lorentz force [11] have to be stronger, or there has to be a larger radius, meaning less curvature, which in turn increases the size.

Another possibility to achieve higher energies, is to use heavier particles. The LHC uses protons, which in contrast to electrons lose less of their energy via synchrotron radiation, when being accelerated by the bending magnets [12].

The LHC itself is a proton-proton collider consisting of two circular beam-pipes with a length of 26.7 km. Within these beam-pipes, proton bunches consisting of around 10^{11} protons are accelerated in opposite directions. There are four interaction points, where the beams cross paths and where the protons can collide with each other. The four big experiments ALICE, ATLAS, CMS and LHCb are built around these interaction points, in order to capture the resulting particles. The design similarities between ATLAS and CMS exist purposely to verify and cross-check results.

One of the goals of the LHC is to generate physics processes beyond the Standard Model [13].

Event rates

Apart from the energy, another important characteristic of an accelerator is its luminosity. Especially when trying to get statistically significant data, as well as when trying to observe rare processes, the amount of collected data, the integrated luminosity, becomes important.

A higher luminosity means more collisions per bunch crossing. As a consequence, the background noise in the collected data, called 'minimum-bias', increases [14]. In the end, there is a trade-off between luminosity and background [14].

The High-Luminosity LHC (HL-LHC) has been approved and will be commencing operations in 2026. It will increase the design luminosity by a factor of five [15]. This can be achieved by reducing the beam size, and the bunch distance and length, and by having bunches with more protons.

The LHC does not consist of only one ring, but is part of a system of pre-accelerators that supplies the LHC with high velocity protons. A schematic view can be seen in Figure 3.1.

The starting point of the acceleration is a bottle of hydrogen gas. First, the hydrogen is stripped of its electrons. The resulting protons are then injected into the LINear ACcelerator LINAC 2 that is close to the bottom in Figure 3.1. At the end of this acceleration, the protons have an energy of 50 MeV.

Following the path in the picture, the next acceleration step happens in the Booster in front of the Proton Synchrotron. This Booster accelerates the protons until they reach an energy of 1.4 GeV.

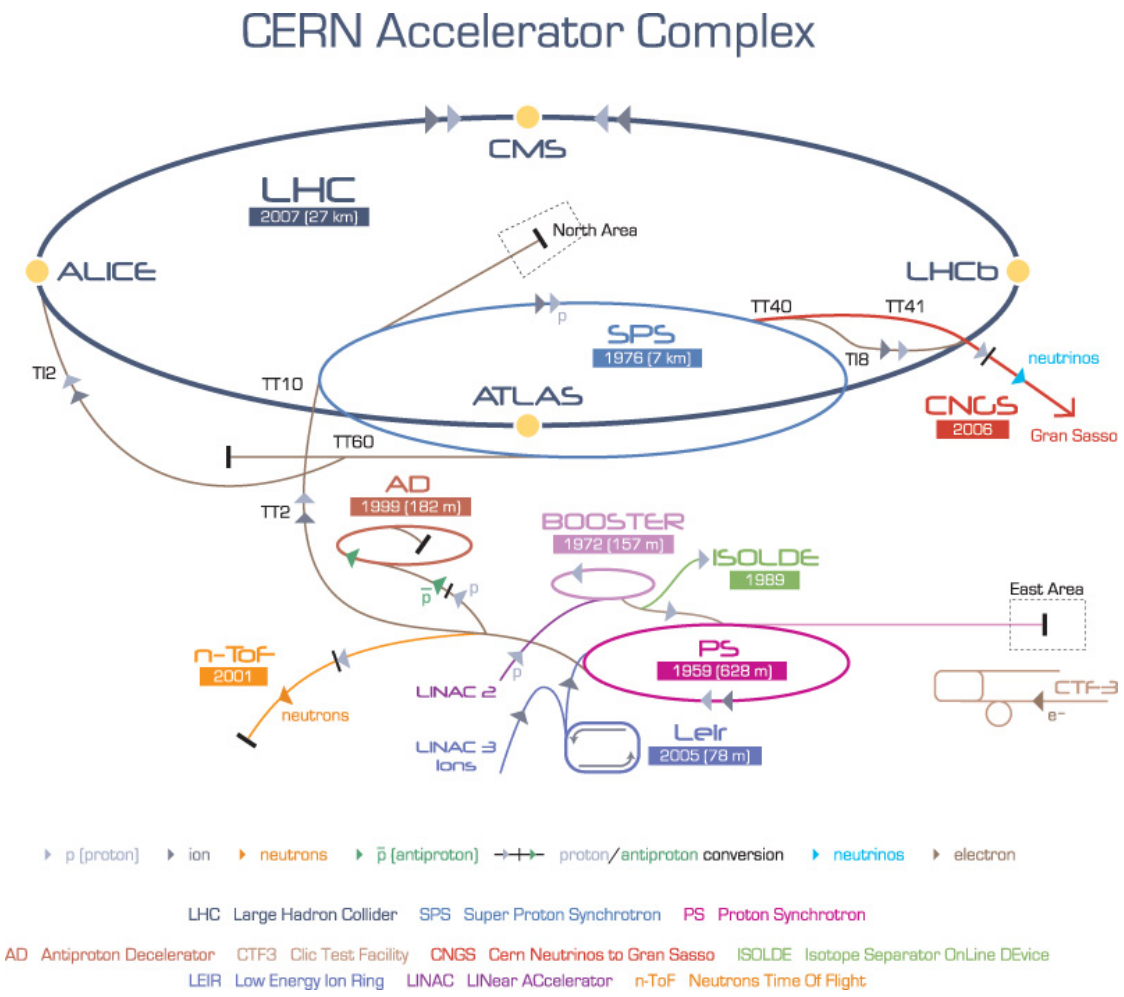

Figure 3.1: Schematic view of the LHC accelerator complex.

From there, they pass into the Proton Synchrotron (PS), which can accelerate the protons to 25 GeV.

Afterwards, the protons are accelerated further in the Super Proton Synchrotron (SPS), that discovered the W and Z bosons. In there, they are accelerated up to 450 GeV, before reaching the final accelerator, the LHC.

The LHC accelerates the protons to their maximum energy of 7 TeV.

It can be said, that the previous generations of accelerators that CERN has, act as pre-accelerators for the newest collider.

3.2 ATLAS

With 46 m in length and 25 m in diameter A Toroidal LHC ApparatuS (ATLAS) [6] is the biggest experiment at the LHC. It is a multiple purpose detector that was designed over a period of 15 years specifically to match the conditions of the LHC. It is built in several layers around the interaction point, as can be seen in Figure 3.2.

3.2.1 Detector components

In order to determine which particles have been created by a collision, without having the possibility to directly measure them, most resulting collision and decay products have to be measured by the detector.

The only exception are neutrinos which only interact via the weak force and therefore pass through matter almost unobstructedly [1]. The ATLAS detector cannot detect them directly, the presence of neutrinos is inferred from the missing transverse energy as well as the missing transverse momentum. There is no favoured direction, so the total sum of all momenta/energies from a collision should add up to zero. The neutrino is therefore detected only indirectly.

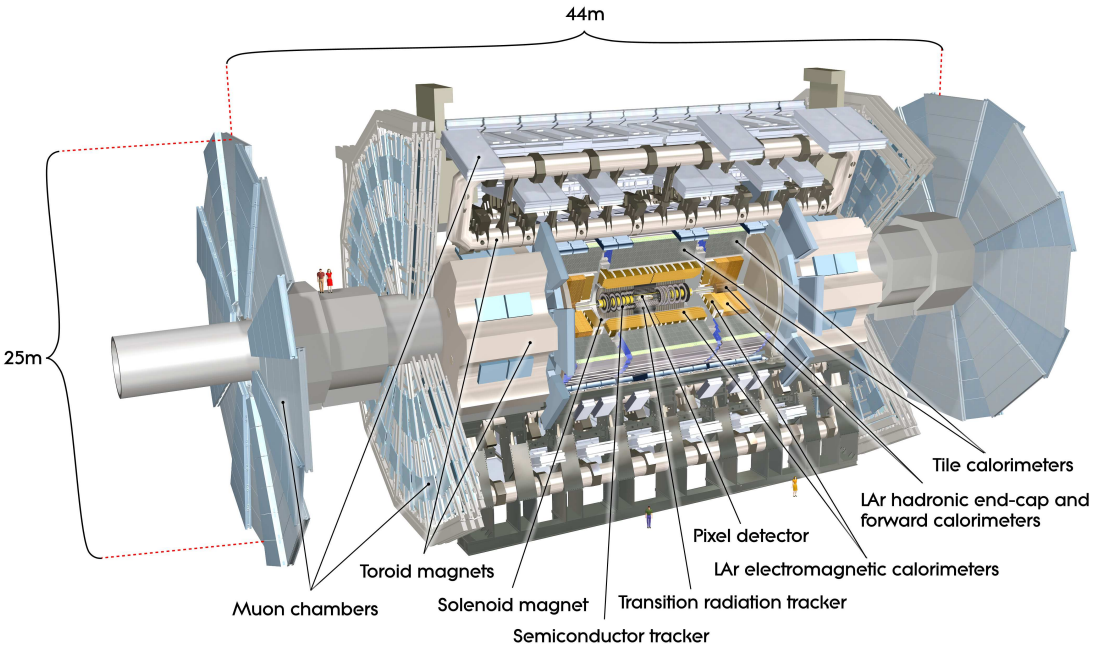

Figure 3.2: Schematic view of the ATLAS detector, taken from [6].

In Figure 3.2 it becomes clear that the ATLAS detector encompasses almost the entire space around the interaction point. The only exception is the beam pipe from and to which the particles come and go.

A large part of the detector consists of magnets. The magnetic field, which the detector is immersed in, helps to identify the momentum and charge of a particle. The Lorentz force bends the track of a charged particle into a circular shape.

The figure illustrates that the detector consists of several different layers. The layers of the detector are built in a way, that low energetic particles which interact strongly with the detector materials are detected first. Less strongly interacting particles can transverse these innermost layers almost unaffectedly.

3.2.2 Inner detector

The Inner Detector (ID) is used for pattern recognition, momentum and vertex measurements, and electron identification. The strength of the magnetic field in the ID is ~ 2 T.

The innermost part is made of pixels and the semiconductor tracking (SCT) detectors, consisting of silicon microstrips [16]. They are constructed in a way that at least four layers of strips and three layers of pixels are crossed by each particle. Further out, the inner detector contains the Transition Radiation Tracker (TRT), which is a straw tube tracker that allows a continuous track-following [16].

3.2.3 Calorimeters

The calorimeters determine the energy of the particles, which is deposited entirely in the calorimeter. From this they generate an output signal that is proportional to the particle energy deposited in the detector. Therefore, calorimeters consist of two layers, namely an absorber and an active material that produces the output signal. There are multiple processes of particles reacting with matter. Especially for different energies, different processes can become more or less influential on the absorption, as can be seen in Figure 3.3.

The particles entering the calorimeter generate a particle shower, that has to be contained in the calorimeter. The absorption strength and the particle energy determine the size of the calorimeter.

In ATLAS, two different kinds of calorimeters are used. The first one is the electromagnetic calorimeter, which can detect electrons and photons. The second type is the hadron calorimeter which detects pions, protons, kaons, and neutrons. How different particles can be distinguished is shown in Figure 3.4.

The types of particles shown leave distinctive signatures within the different detector components/calorimeters. A further differentiation between particles with similar signatures, such as protons and kaons can be done, by looking at additional parameters, such as the ionisation energy loss.

Electromagnetic calorimeters

In ATLAS, the electromagnetic calorimeter uses Liquid Argon (LAr) as its active material [18]. Lead is chosen as an absorber. The size is over 22 radiation lengths in order to

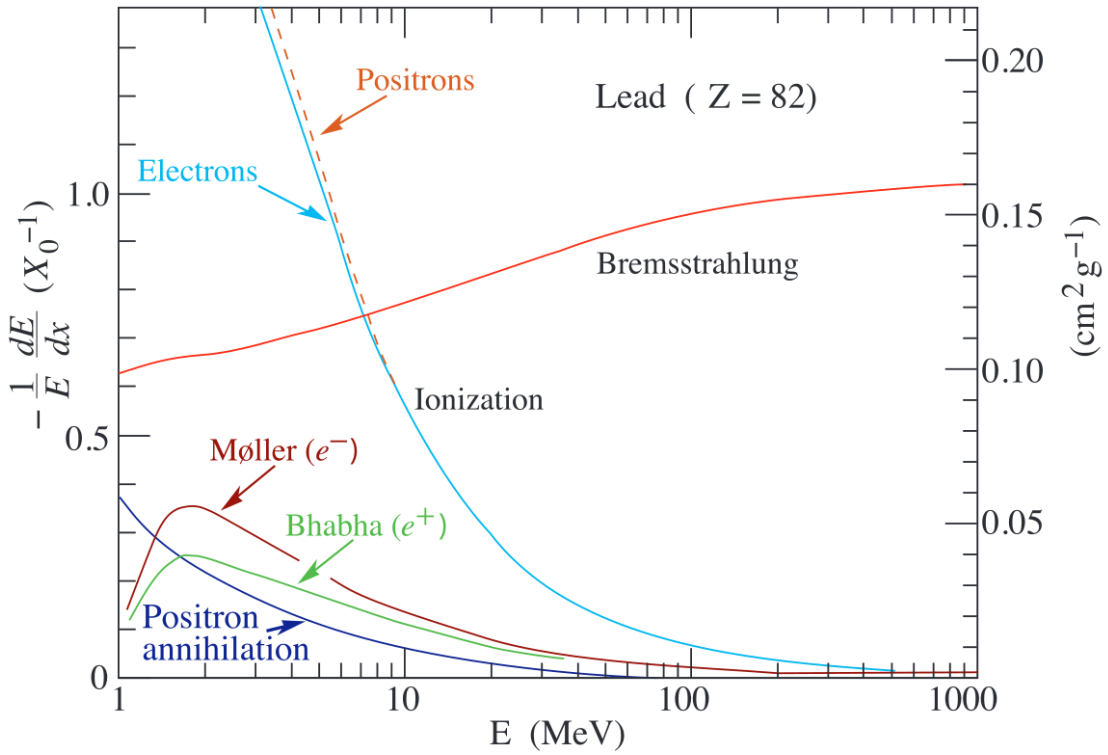

Figure 3.3: Fractional energy loss per radiation length in lead as a function of electron or positron energy. Electron and positron scattering is considered as ionisation when the energy loss per collision is below 0.255 MeV, and as Møller (Bhabha) scattering when it is above. X_0 is the absorption length and E the particle energy. (Adapted from Fig. 3.2 from [17]. Messel and Crawford use $X_0(Pb) = 5.82$ g/cm^2, here the figure reflects the value given in the Table of Atomic and Nuclear Properties of Materials ($X_0(Pb) = 6.37$ g/cm^2) [1].)

prevent electrons and photons from reaching the next detector layer, where they might not be detected.

Hadronic calorimeters

The hadronic calorimeters consist of LAr and Tile calorimeters. The LAr calorimeter uses tungsten and copper as absorbers. In the Tile calorimeter, steel is used as the absorber and scintillating tiles made out of plastic as the active material [19].

3.2.4 Muon spectrometer

The muon spectrometer consist of four parts, which detect muons without absorbing them completely. The Monitored Drift Tubes (MDT) measure the curves of the tracks

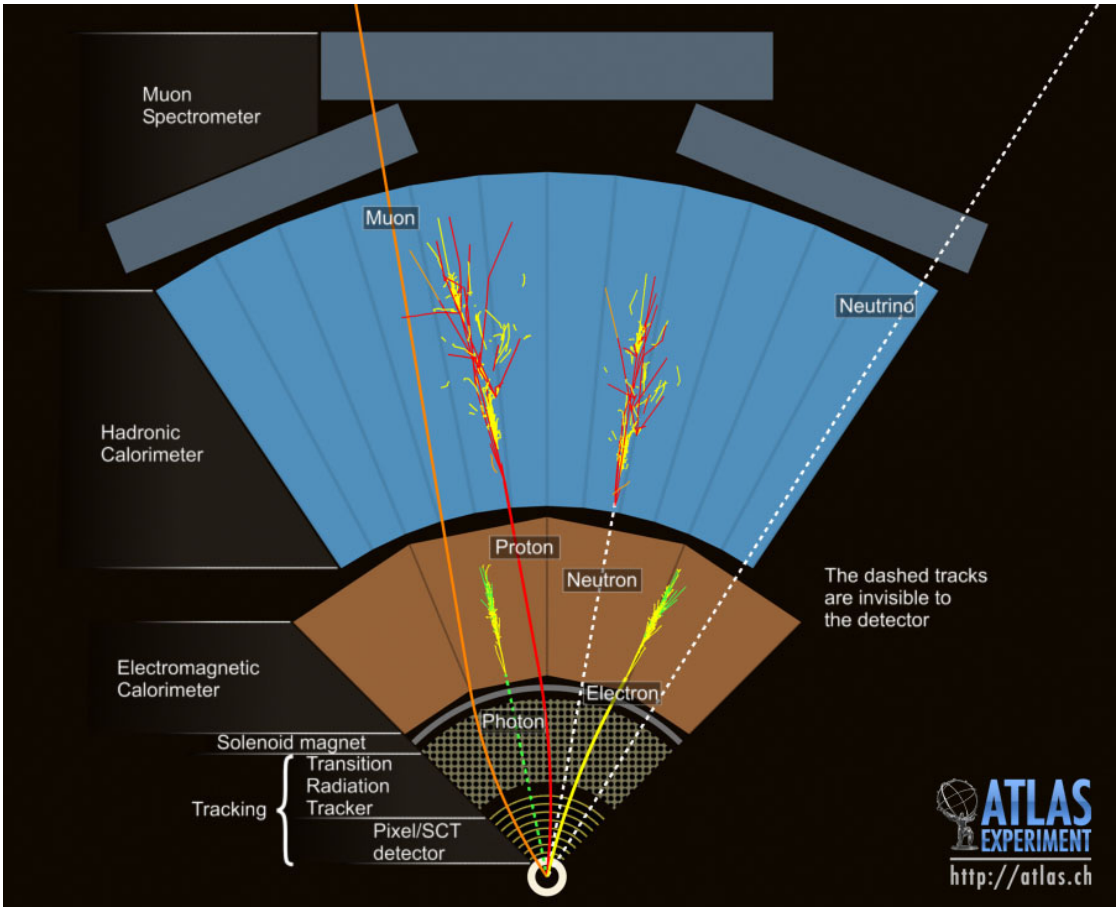

Figure 3.4: Schematic representation of different particle tracks within the ATLAS detector. Electrically neutral particles are not detected in some parts of the detector, this is represented by a dashed line.

[20]. In addition, Cathode Strip Chambers track the position in the end caps [20]. The Thin Gap and Resistive Plate Chambers provide the required trigger information, see Subsection 3.2.5 [20].

3.2.5 Trigger and data acquisition

The Trigger and Data AcQuisition (TDAQ) systems are needed in order to collect all the relevant data of the particle collisions. While running, the detector parts that have been described above, deliver a constant stream of event data. However, due to monetary and technological constraints, it is not possible to process and store every collision. In order to reduce the amount of data that is coming out of the detector, the Trigger system is in place. It reduces the data rate from the design rate of 40 MHz, to a few hundred Hertz, [21] [22].

This large reduction is possible, because many bunch crossings do not contain physics analysis relevant collisions. These can be, for example, scattered protons or already well known processes, that do not have to be examined again.

The trigger system consists of several levels, that perform increasingly complex decisions on whether to store the event data.

The level 1 trigger reduces the incoming event data to around 100 kHz [22]. The high trigger speed is achieved, by looking at only a subset of the data from the detector components. This high level filtering is hardware based.

The level 2 trigger analyses further regions of interest, that were indicated by the level 1 trigger. The events that go from the level 2 trigger to the next trigger level, have been reduced to a frequency of around 1 kHz [22].

The level 3 trigger analyses the full event data and reduces the event rate to a final frequency of a few hundred Hertz.

CHAPTER 4

LHC offline computing

"I do not fear computers. I fear the lack of them." - Isaac Asimov

In the title of this chapter, a distinction between two different computing concepts is hinted at - online and offline. Offline computing encompasses all data processing from the raw-data input buffer to the result of a physics analysis, whereas online computing encompasses the computing up to that point, meaning triggering and data acquisition (see Subsection 3.2.5). Within this thesis, computing refers to the offline computing unless stated otherwise.

The computing at the LHC faces a drastic increase in required resources. It is difficult to predict what the exact needs will be, as they highly depend on the luminosity, the pileup and the LHC performance. The LHC performance can be characterised by the fraction of time within a year in which data is taken. The HL-LHC is estimated to increase the resource requirements manifoldly. The predictions for the year 2016 for example, were wrong, because the LHC performed above expectations, with a luminosity above the design level and a very high availability. In the end, additional computing resources, especially storage, had to be made available.

Moore's law [23] successfully predicted the technological evolution of processors. Indeed, not only have processors been improving, but most components of modern computers, like disks, RAMs, networks, etc. have become better and faster over time [24]. This technological evolution is one of the driving reasons why, for the same budget, better infrastructure can be bought at a later point in time.

In terms of computing, the increased luminosity means that there will be an estimated factor of ~12 more data, and an additional CPU power requirement of the factor of ~60,

compared to 2016[1]. In contrast, the budget for computing is flat. With the current technological growth of around 20% per year, the additional requirement for computing power boils down to a factor of ~10.

In order to close the gap between estimated and available computing resources, either the software or the hardware have to become better. In this thesis, only infrastructure improvements are considered, whereas software improvements, such as more efficient algorithms, are ignored. The only overlap between the domains would be the scheduling. It is, however, a combination of workflow (see Chapter 5) and hardware configurations (see Chapter 8) and therefore falls into the infrastructure improvement domain. This scheduling technique is an example of the positive impact of the evolving technology of Cloud computing described in Section 4.2.

In the end, there will most likely not be one miraculous solution that can mitigate the discrepancy. It is expected that a combination of many smaller improvements, like the introduction of Cloud computing, will solve the issue. The problem with Cloud computing so far is, that the gains are difficult to quantify as there are many factors that have to be considered. In this thesis, the issue was solved by developing and applying a model, that is able to directly compare different Cloud computing offers with the Grid computing concept, see Chapters 6 and 7.

4.1 Distributed and Grid computing

A key concept for the LHC data processing is distributed computing. Distributed computing consists of a system of dispersed computers and computing centres, that are interconnected through a network and unified by a high-level system with transparent components [25].

From a physicist's point of view, it does not matter where in the world the data pre-processing happens. The location where a physicist's analysis is computed is also of no importance to the physicist, as long as the correct results are delivered within a reasonable time. However, the main reasons why the computing was distributed, were of a sociological and not a technical nature. Namely, the fact that computing investments are local. This means even small institutes make contributions to the computing through their clusters, which would not have happened otherwise. In addition, there are benefits in providing training for students, while being able to leverage the resources for other local uses [26].

A more in-depth look reveals that the scenario profiting the most from distributed computing, is the one in which complex problems can be broken down into several smaller problems. This applies to High Energy Physics (HEP) where, as we have seen before, each detector event is a small sub-problem. In practice, multiple events are computed in one process or workflow and multiple workflows are then computed in parallel. This

[1] According to the presentation of Ian Bird at the 2016 WLCG computing workshop in San Francisco.

parallel computing is equivalent to an increase in computing power, which results in a faster solution. An example would be the processing duration of 10000 events on one computer. It can be roughly cut in half by having a computer that is twice as fast, or by processing the events on two computers of the same speed. Easy scaling by adding computers, is one of the benefits of distributed computing.

Another benefit is that resources can be shared between different groups around the globe, achieving less idle CPU time. This also adds a kind of reliability to the distributed computing as there is no single point of failure. Finally, it might be cheaper than buying one powerful computer. More on the benefits, challenges, disadvantages and implementations can be found in Section 4.4.

4.2 Cloud computing

"Cloud computing is the third wave of the digital revolution." - Lowell McAdam

Cloud computing might be a solution to the impeding resource shortage. Since the emergence of Cloud computing from different commercial companies on the open market, the prices have been falling. The most extreme cuts in prices happened in the early years, between 2011 and 2015, when the big providers cut their prices roughly in half. This development has slowed down and changed since, by providers offering better hardware for the same prices for example.

One factor is of course the technological evolution that was already mentioned earlier. A white paper by ESG Labs, commissioned by Google, states that Google will pass on price reductions from technology-driven advancements to all customers [27]. A big Cloud company can build computing centres that are multiple times bigger than what each individual customer would have to build. This bigger scale reduces the overall operational and even infrastructural costs, making Cloud computing more profitable for both sides.

Another factor is the competition on the market between the individual providers. Since many vendors are selling similar products and competing with their prices, the Cloud market could almost be considered to be in "perfect competition"[2], as described in [28]. This has downsides, especially for the customer. In order to undercut the competition, the offer of a provider only has to seem as if it is better. This leaves room for Cloud providers to use their freedom to deceive customers, by for example making their Cloud seem cheaper by applying hidden fees. Another possibility is that the provider supplies a lower computing performance, which is difficult to figure out from the customer side. One possible solution to this problem could be a universal Cloud service certification, that would make the different offers comparable, as suggested in [29]. The real cost and performance of Cloud providers are evaluated in Chapter 7. A different prediction of the market behaviour is that the pricing pattern follows longer periods of stable prices, with price wars between providers at certain points [30].

[2]The opposite of a monopoly.

4.2.1 Concept

The idea of a huge network of interlinked computers exists since as far back as the 1950s and 1960s[3]. Selling computing as a commodity is the next logical addition to this picture, but only with the growth of the global networking infrastructure did it become achievable, especially for individual users.

The real motors behind the sudden emergence of Cloud computing were big online companies. After realising the extent of their unused infrastructure, which is designed to be able to handle peaks in demand, Cloud computing must have seemed like a lucrative solution. Especially, since the hardware had already been purchased and was already constantly running.

Cloud computing is a broad field, and the boundaries between what is considered Cloud computing, and what is not, are blurry. Even years after the emergence of Cloud computing, there was no clear-cut definition. A summary was attempted in 2008, that can be paraphrased as: Clouds are many easy-to-use and -access virtualised resources [31]. These are adjustable and configurable to a varying workload on a pay-as-you-go model.

The NIST definition of Cloud computing, that appeared later, is more specific and is therefore the one adopted in this thesis. It is not too different from what was previously found. "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [32].

The main characteristics according to this definition are on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service [32]. Even though the Cloud providers that are discussed later on fulfil these criteria, there will be limitations. These can be found especially in the rapid provisioning and releasing of resources. The NIST definition divides Cloud computing into three service models.

The first one, Software as a Service (SaaS), means that a customer can run a provided application (on a provided Cloud infrastructure).
Platform as a Service (PaaS) is the second service model, which provides more freedom to the customer, to deploy their own applications.
These two models are not viable for most computing in high energy physics, as a high level of freedom to control the underlying operating system and storage is required.
This is achieved via the Infrastructure as a Service (IaaS) model [32]. In this thesis when not explicitly stated otherwise, Cloud computing refers to IaaS.

It makes sense to distinguish between three different kinds of Clouds: private, public and hybrid.

The private Cloud is deployed within the organisation, reducing the security exposure and legal ramifications that may result from the usage of outside resources.

[3] For example in literature, short story "Answer" by Fredric Brown

In a public Cloud all the infrastructure is located at the third party provider and provisioning happens over the Wide Area Network (WAN).
A hybrid Cloud combines the two previous concepts, by having Cloud resources on- and off-premise, combining them through common technology [33] [34] [32]. A popular use case is called bursting [35], meaning to absorb peaks in resource demands. An example of an online shop using bursting can be found in Subsection 4.2.2. The workload in HEP could benefit from bursting, as it follows a pattern that has peaks in demands, especially before high-profile physics conferences. In that time frame, everyone wants to include the latest and most recent results in their presentations.

The NIST definition does not explicitly mention the model in which Cloud infrastructure is procured statically, meaning for example purchasing X amount of machines over a period of Y months. It technically falls within the IaaS model. The reason it is mentioned here, is because of certain boundary conditions with procurements within some organisations, like CERN. First of all, for these organisations the only possibility to procure something is for a fixed amount of funds, so the pay-as-you-go scenario is not possible. Furthermore, for procurements above a certain threshold it is mandatory to have an open tendering phase, during which companies can bid for the contract. In order to compare offers, the whole procurement has to be well defined in advance. On the upside, studies have shown that these kinds of procurements are more cost effective, because the Cloud providers can also plan their resources better. The Amazon prices for these type of resources for example can be significantly cheaper. Amazon states that a discount of up to 75% is possible by purchasing reserved instances instead of the on-demand ones. Other providers also offer discounts and incentives to commit to a certain amount of resources beforehand, which will be discussed in further detail in Subsection 4.2.2.

This broad Cloud computing definition corresponds to offers from companies such as Amazon, Google, Microsoft, etc. There exist differences between what and how companies offer. As a customer the most important thing is to understand these differences and how they impact the performance and overall cost. There are some downsides, since the hardware that one receives from a provider is more or less a black box. One of these downsides is drops in performance due to overallocated resources. An analogy would be the overbooking of an air plane. As long as there is a sufficient amount of customers that do not make the flight, the airline increases its income. If too many passengers show up, there will be negative consequences.

The overallocation is possible due to the virtualisation of physical machines. One example for this is hyperthreading, where two logical processors share one physical processor. Cloud providers could take this even further and overcommit much more. Up to a certain degree this makes sense.

Even though customers procure Cloud computing on a per-usage model, they still do not use 100% of what they buy. There are inefficiencies within the applications, such as the peaky demand or I/O intensive applications. In addition, suboptimal usage can

stem from overheads and possibly the procurement of more resources than needed. An example would be a customer that cannot accurately predict their own workloads. Overcommittment strategies have been studied, see for example [36], where most workflows used less than 20% of the available CPU capacity, making overcommittment viable even when large safety margins are applied.

There are even examples of Cloud providers that do not own hardware themselves, but are instead procuring their hardware from another Cloud provider. Dropbox is an example of such a provider.

4.2.2 Pricing

One of the big advantages of Cloud computing is the pricing model. The prices that the providers publish are given as X euros per hour per instance. Therefore the cost of one server for a million hours is the same as the cost for a million servers for one hour. In practice, the infrastructure of a Cloud provider is not infinite, so this may not hold on large scales. For highly parallelisable workflows this becomes very attractive as the results can be returned in an instant at no additional cost.

Conversely, the Cloud provider benefits from the economy of scale. This effect can be visualised by the fact that the personnel needed to run bigger clusters scales at less than a 1:1 ratio. This means that doubling the size of the cluster needs fewer than twice the employees. Also the unit price for hardware, for example, drops for a larger bulk order.

In the early stages however, it was a result of companies renting out their spare resources. An example for companies with unused resources are online shops, whose needs in the period leading up to Christmas are many times than what is needed during the year. In order to not lose business during these profitable times, their infrastructure must be able to handle these circumstances. As a consequence these companies have a computing capacity that is many times larger than what they need outside of these rarely occurring peak scenarios. This spare capacity lies idle most of the time.

The reason why Cloud computing is attractive from the sellers perspective should now be clear. The same example as above can be used to explain why it can make sense to acquire Cloud resources for a buyer. An online shop that is not very big cannot afford to invest much money in its infrastructure just to be able to deal with the mentioned peaks. It would therefore lose business, were it not for Cloud computing. Now the shop can buy some on-demand resources during peaks and discard them afterwards, without having to invest heavily in otherwise unneeded infrastructure. Of course these two examples do not work together as both stores would have computing troubles during the same time, but Cloud providers and customers are not limited to online shops.

Cloud pricing is on a downward trend as mentioned at the beginning of this Section 4.2. Due to the complex pricing system and the large amount of offers, it is difficult to quantify just how fast this trend is. Indeed, it is difficult to estimate how much a customer would pay on each of the providers and which Cloud offer would be the best/cheapest. There are several tens of thousands of price points per provider. In an

article on InfoWorld, only a small subset of price points for each provider are compared, as a comparison between providers is close to impossible otherwise. Even after narrowing these price points down, there is not one provider that wins out over the others, as the use case strongly influences the result. This is exemplary for the challenge that has to be tackled when comparing the Cloud to the WLCG.

An estimation of the price developments can be seen in Figures 4.1 and 4.2. The first figure indicates that the average annual price reductions lie at around 10%, over the last eight years. This is less than would be expected from Moore's law, but a Cloud site also has other costs than only hardware [37].

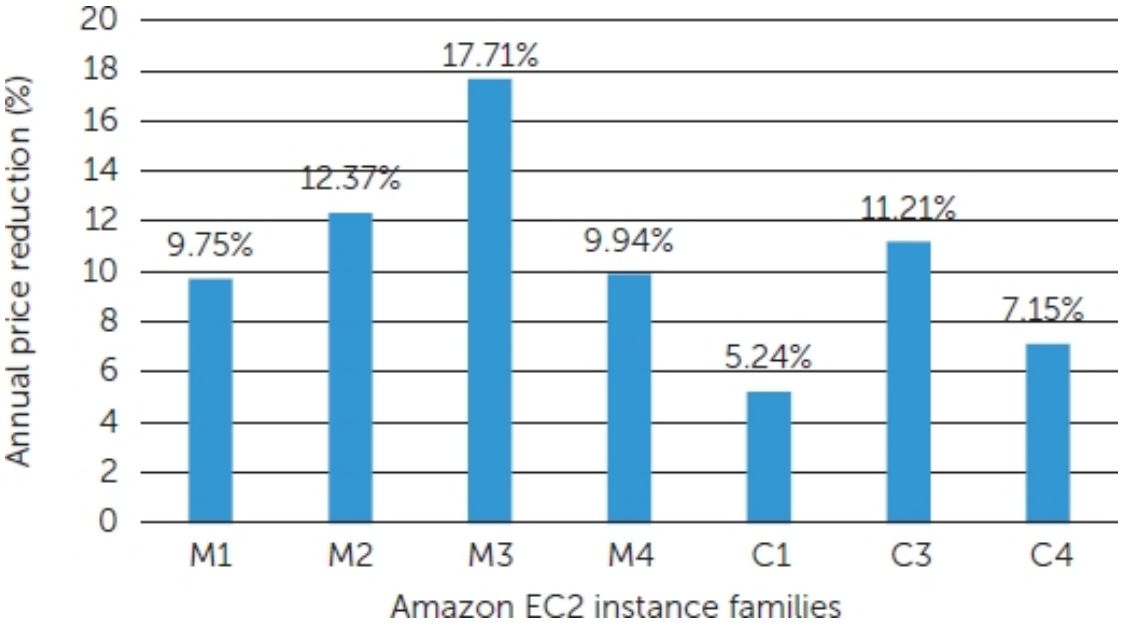

Figure 4.1: EC2 price reductions, with the on-demand payment model. Annual price reduction since their public release. Source: [37]

In Figure 4.2, the history of these price drops can be seen. It becomes apparent that the reductions are not a smooth process, but that they consist of jumps that are triggered by, for example, market competition. The biggest drop happened after Google lowered their fees significantly [37]. Another observation that can be made is that the downwards trend has reduced. The Cloud price development becomes important later in the thesis, when evaluating the viability of moving to the Cloud, see Chapter 9.

The previous figures showed the different kinds of infrastructures that are on offer, but there is more to the pricing complexity. The simple choice of operating system, for example, can change the pricing structures, as different operating systems are priced differently. In addition, the same operating system costs differently depending on the provider. More impactful factors are the various discounts that the Cloud providers offer, which can lead to a price reduction of up to 75%. Another factor is the geographical

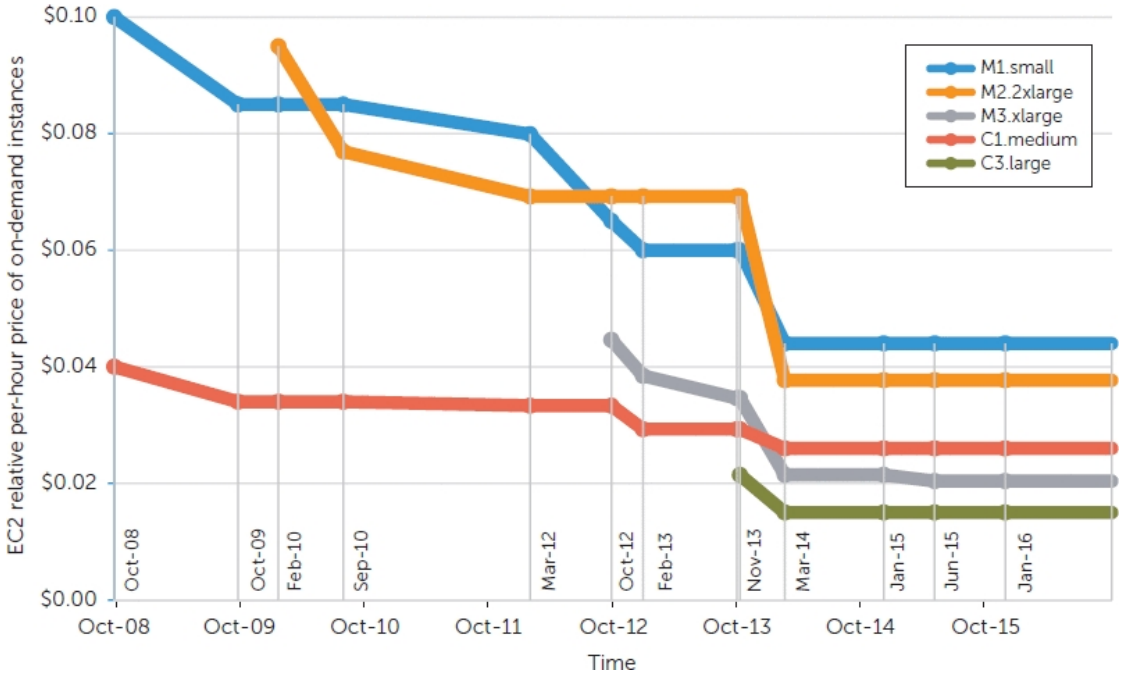

Figure 4.2: The Amazon EC2 historical price development. Displayed are the prices per hour, gauged by the EC2 Compute Unit. This Unit is a relative indicator of the compute power. Source: [37]

region, in which the data centres are located, leading to a difference in pricing of up to 50%. In the end, every use case has to be evaluated separately.

Another substantial cost factor that a potential Cloud customer should not neglect is inefficiencies. In an article published on Rightscale, it is stated that ten billion US Dollars were wasted on unused Cloud resources in the year 2017 alone. Two of the biggest sources of this waste are named as overprovisioning of instances and idle instances. This gives an indication that inefficiencies are a general, large scale problem. Independent of these numbers, idle CPUs or even virtual machines (VMs) can also be observed in the CERN data centre. These inefficiencies are causing providers to use optimisations, such as overcommitting. The overcommitting concept is used later in this thesis, see Chapter 8. The consequences of these optimisations can be drops in performance, as well as fluctuating performance, which is examined in Chapter 7.

4.2.3 Storage

The storage plays an important role, as it has a large impact on the overall cost and personnel requirement of a computing centre. Several factors have to be considered, the most important one being the impact of the storage on the overall processing time.

Whenever a CPU is idle because it is waiting for data input, resources, which in this case can be translated directly into money, are wasted. The CPU waiting time depends on the speed of the storage as well as on the distance to the storage.

In order to store data, there are two possibilities. The first one is to procure storage space within the Cloud. This is the most intuitive approach as it mimics the current scenario of Grid sites, which provide internal storage space. The benefits are for example that the required input data of workflows can be placed close to the CPUs. This leads to a reduction in latency and typically the bandwidth is higher within a site than across different sites. In the end this results in a faster workflow execution.

Furthermore, having storage available offers the possibility to store intermediate workflow products. More importantly, in case it is not possible to directly write to the remote storage, for example when there is little-to-no bandwidth available, the workflow outputs can be stored inside the Cloud and the VM can continue processing data.

In addition, some workflows have common input data. For those, every subsequent workflow can exploit an existing data locality. The benefit in speed is achieved without increasing the required storage space, and therefore without increasing the cost. The WAN traffic would also be greatly reduced, as the data have to be transferred only once. In the case of ATLAS, an example that follows this usage pattern would be the pileup, see Subsection 5.3.3.

Conversely, there are the downsides, which for the most part manifest themselves in the cost. They therefore highly depend on the cost models of the providers. The pay-as-you-go model for storage means, one usually pays for the amount, duration, and type of used storage space. Sometimes this also includes data egress, ingress, and/or Input/Output Operations per Seconds (IOPS). Depending on the use case, the cost model of the provider can make the Cloud storage procurement completely unfeasible. An example would be analysis data, see Section 5.4, that are stored in a Cloud that charges for IOPS. A popular analysis dataset is accessed around the clock. The cost of this scenario would by far outweigh the benefit of the gain in speed.

The second possibility is to not make use of the storage offers of the Cloud providers. Instead, the required inputs are either copied to the local disk of the VMs at the beginning of each workflow or read on the fly, during the workflow execution. These capabilities are already in place and actively used by multiple LHC experiments. An example of the latter would be the “Any Data, Anytime, Anywhere (AAA)” service of CMS. It is an XRootD (see Subsection 4.5.1) service that enables CMS data to be read remotely. This helps to distribute the CMS workload to sites that do not have the necessary input data.

The money that would have been spent on storage can then be invested in different aspects of the infrastructure. An example would be to invest in better bandwidth and receive a similar performance for a reduced cost, compared to investing in storage. Another possibility would be to acquire more computing power, or a combination of the

two.

In the case of workflows that read different input data, such as raw data reconstruction, the input has to be transferred into the Cloud exactly once. In terms of WAN transfers, this scenario would not benefit from additional storage within the Cloud.

The downside to not having data stored within the Cloud is that the bandwidth may become a bottleneck. Generally speaking, for too large input datasets, bandwidths today do not scale sufficiently. Too large means not only the size, but also the speed at which the data are processed, since a faster data processing requires more input data in the same amount of time.

This can be illustrated by the Amazon "Snowmobile", a service that transfers up to 100 PB of data by physically transporting storage media on a large truck from the customer's data centre to the Cloud data centre. Other experiments which highlight the WAN's limitations, tested the data transfer via the internet versus a pigeon carrying a USB stick. In that experiment, the pigeons came out as viable alternatives [38].

Apart from the data that is under analysis, the LHC experiments also have certain data which is archived. This can be, for example, the raw detector data. Cloud providers also offer storage space for archival purposes. This storage space is usually much cheaper, but accessing the data may come at a cost, usually in terms of time as well as money[4].

A risk of having data archived inside the Cloud is the vendor or data lock-in, which has to be avoided [39] [40]. It means a customer is depending on a single vendor and changing vendors would be accompanied by substantial cost. One factor may be incompatible storage technologies between the providers. It can also be the case when all data has been transferred (in whichever way) to one provider and the effort of switching providers means having to transfer the data again. Especially when time constraints and also the cost of transfer can make this nearly impossible, resulting in a situation where the vendor can abuse this position of power by raising prices.

Object storage

In contrast to the WLCG, most Cloud providers use object storage as their storage technology of choice. Object storage treats data as storage-objects in a flat, unstructured hierarchy. Objects are accessed via RESTful interfaces, which are simpler than regular file systems, through the unique identifier that is attached to them. In addition they come with a flexible amount of metadata. The object store validates user credentials that are attached to each operation. Access control therefore happens on a per-object basis [41]. The advantages for this method are scalability and cost-effectiveness, which makes it attractive for Cloud providers. They can simply attach more and more storage devices in order to scale up [42].

Conversely, object storage comes with a decrease in performance [43]. In addition, the

[4] An example would be Amazon Glacier.

REST-based calls of the object storage have to be integrated with the existing WLCG storage infrastructure.

4.2.4 Security, safety and integrity

Studies have shown, that security is one of the biggest areas of concern for companies regarding Cloud computing [44].

There is a wide variety of problems that go along with computing and storing data at a third party site. Most of these issues are a result of the fact that Cloud computing customers effectively give up the control over their own data. However, in the case of LHC data processing, some risks that would affect other customers only play a minor role.

The LHC data, although it is private, is not very confidential or sensitive, and there are no explicit laws regarding its handling and security. Private companies may run into trouble with the multi-national nature of Cloud providers, that have data centres around the Globe. From the fact that the WLCG also spans the Globe, it can already be seen that no national laws prohibit the distribution of the data, as might be the case for e.g. hospitals.

Outside of physics, the LHC data is not valuable. Considering also the large amount of data, it is not attractive for thieves to try and steal the data.

In case the Cloud provider loses data, the most that will be lost are some results that were temporarily stored within the Cloud. No permanent Cloud storage would be used for archival purposes, the original data will rest within the WLCG.

Despite these mitigating factors, there are still risks associated with using Cloud providers. The immediate concern is whether the Cloud provider can be trusted. Dishonest providers or system administrators may violate the privacy by stealing user credentials, making usage of their privileged role. For example, by directly accessing the memory of a VM as described in [45] or by accessing the private data in the Cloud storage.

Another issue may be, that the Cloud provider does dishonest computation, meaning instead of executing a CPU intensive computation, the provider could simply give a wrong result and thereby save resources [46] [44]. Apart from malicious intent, the result may simply be wrong [46] [44]. This could be due to, for example, hardware failures or software bugs over which the customer has no control or knowledge. In the worst case, this could falsify many physics analysis results.

Data integrity can suffer from similar problems, such as hardware failures. The consequences are similar, therefore the data integrity has to be ensured by the Cloud provider.

Furthermore, the security of the data rests with the Cloud provider. In addition to the regular threats, the attack surface is much higher as hacks may be attempted from within. An attacker may even execute malicious code on the same hardware as the customer and the Cloud provider has to make sure that these cross-VM attacks cannot threaten any customers [47]. Prominent recent examples, where collocation on the same hardware can be an issue, are Spectre [48] and Meltdown [49]. These are hardware

exploits prevalent in almost all modern (micro-)processors. The associated risk is to lose resources and/or user credentials as well as having results manipulated.

Trust in the provider is also required considering the pay-as-you go model. Combined with a highly dynamic environment, this can make it difficult to accurately charge for the services used. From the customers point of view, it is difficult to verify whether the invoice represents reality or not, especially considering the high complexity of the cost models.

Many trust related issues are handled in the Service Level Agreement (SLA), a binding contract or policy of a Cloud provider, where infrastructure as well as, for example, privacy and security guarantees are given to the customer.

In general there is access control over all data/VMs in the Cloud [45].

Another proposed solution are external audits that a Cloud provider should undergo [44].

4.2.5 Availability

A very important factor when using Clouds is the availability. After outsourcing data or computational workload to the Cloud, a customer wants to be sure that the data or the result of the computation can be accessible when needed. Depending on how time critical these processes are, it can be catastrophic for a customer if the Cloud is down at the wrong time. In general, Cloud providers sell their services together with a guaranteed availability level. However, it could be that a Cloud provider knows that its infrastructure is not built to actually reach this guaranteed uptime [28]. The benefits of promising this too high of a guarantee may outweigh the penalties that the provider will incur, when the uptime is not reached. For a customer, who is depending on the guarantee, this business model can have very bad consequences as the compensation may be only a small fraction of the lost business. For a WLCG site, it could lead to a loss in reputation and have consequences to its funding if it is not able to provide the availability it guaranteed in its SLA.

The same may happen if the provider suddenly goes bankrupt and all resources are lost.

Apart from the Cloud provider, there are outside factors that can compromise a customer. As with any data centre, natural disasters such as floods, earthquakes, lightnings, etc., can lead to data loss, as can be seen from the loss of data at Google due to lightning strikes[5].

Further factors, related to law enforcement are the behaviour of other customers on the same Cloud, which can have an impact on all customers, so called “fate-sharing” [50]. For example Spamhaus blacklisted many EC2 IP addresses after a spammer abused the Cloud this way. Afterwards, even legitimate users could not send emails via EC2 anymore [51]. Another example was the FBI raid during which all hardware at a data

[5] Google publishes information on incidents that happened on their platform.

centre in Texas was confiscated, due to the suspected malicious activity of one customer, disrupting the business of many innocent customers [51].

Some example solutions include the fact that Cloud providers allow customers to choose where[6] they want to store/process their data. This can eliminate some issues with law enforcement agencies and limit the exposure to natural disasters. Data back-ups across several data centres decreases the risk of data loss. Most importantly, a customer should choose its provider carefully. This can already avoid many problems. One of these problems that can thereby be avoided with a high probability, is the provider going bankrupt.

4.3 Grid Computing

The term Grid computing is being used in analogy to the power grid, which provided a standardised, dependable and transparent access to electricity [52] [53]. Grid computing is one form of distributed computing. It is the solution to the resource sharing needs of multiple interest groups. These interest groups, that define the access conditions of the users/institutes are called virtual organisations (VOs). The ATLAS collaboration, as well as the other LHC experiments, form VOs. The Grid gives access to computers, software, data, and other resources to the VOs [54]. Sometimes data grids and computational grids are distinguished, in this thesis the Grid refers to a combination of the two.

The Grid handles the entire resource sharing strategies, including the access control, the resource quotas and the placement of workloads. This is handled in a flexible way, as users, resources and VOs can join or leave at any time. It includes management tools for the computational, storage and network resources as well as for code repositories and catalogues [54].

Some of the benefits of the Grid are, amongst others, that collaborators achieve a better sharing and cooperation with results being immediately available for everyone. Furthermore, the Grid can achieve a better utilisation of idle resources, as well as provide high computing power for individual use cases that need it at certain intervals [55].

In general, the Grid can be a heterogeneous mixture of different hardware and software components. These are centralised and accessed through a uniform interface [55].

4.4 WLCG

The WLCG focuses on High Throughput Computing (HTC), meaning it tries to exploit its resources efficiently on a long term basis [56]. In the end, as many physics events as possible should have been processed, in contrast to, for example, High Performance Computing (HPC), where the emphasis lies on how many floating point operations per second (FLOPS) can be performed. Since the WLCG workflows are highly parallelisable

[6] Geographical location of the data centres, usually specified in regions or zones.

and oftentimes I/O intensive, it is not important to simply have the most powerful machines. Factors such as the layout, organisation or the scheduling have to be considered, which will be explained in detail in the following sections.

4.4.1 Concept and purpose

The Worldwide LHC Computing Grid is one of the biggest Grids in the world, constructed for the purpose of computing the physics data generated by the LHC [57]. It consists of around 170 interconnected computing centres in 42 different countries around the world. Its sole purpose is to provide computing power and storage space for the LHC experiments and the physicists analysing the LHC data.

Tier structure

The WLCG computing sites are split into different categories, called "Tiers". These were originally intended to distinguish more powerful and better-connected sites that handle different tasks from less powerful ones. Higher Tiered sites have also stricter requirements for the availability. The network connections were initially chosen according to the hierarchical Models of Networked Analysis at Regional Centres for LHC Experiments (MONARC) model. It assumed that the network bandwidth would be the scarcest resource [58] [59]. From there, the concept evolved to the current situation, where the tasks get increasingly distributed to all sites and the computing power does not correspond to the classification anymore [60].

What remained the same, is the Tier 0 centre at CERN, which is still the centre of the whole WLCG. Due to the short distance between the Tier 0 centre and the ATLAS detector, all raw data is stored and reconstructed there, see Section 5.3. The raw and reconstructed data is then distributed amongst the other Grid sites, which was originally only the Tier 1s. The raw data is distributed in order to have a replica as a fail-safe and to speed up reprocessing campaigns (see Subsection 5.3.6), whereas the reconstructed data can be used by physicists for their analyses, see Section 5.4.

There are 13 Tier 1 sites distributed around the world. Their responsibility is to further distribute the reconstructed data to the Tier 2 sites. In addition they perform some raw data reprocessing and store simulation results (see Section 5.2) from the Tier 2s. The Tier 1s are connected to CERN via the LHC Optical Private Network (LHCOPN), which runs on high-bandwidth optical fibre links [61].

The many Tier 2 sites are located at universities and research institutes, and their tasks are to process simulations, reconstruction of simulated data, and data analysis. The II. Institute of Physics at the Georg-August-University of Göttingen operates a Tier 2 centre called "GoeGrid", located at the Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG).

At the lowest level are the small institute clusters which are called Tier 3s, even though they are not formally engaged with the WLCG.

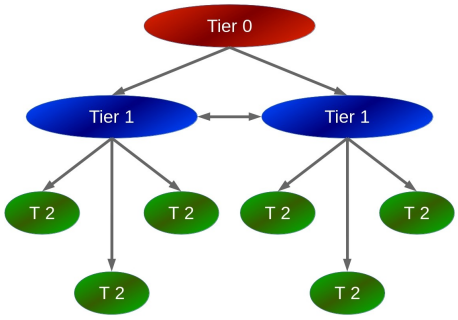

Figure 4.3: MONARCH model.

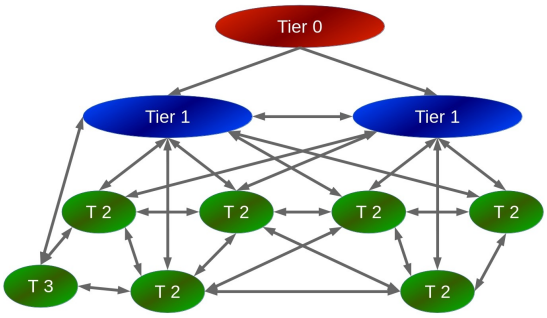

Figure 4.4: Evolution from MONARCH.

Figure 4.3 gives a schematic view of the original Tier hierarchy and network interconnections according to the monarch model. Figure 4.4 shows the evolution, giving an overview of the above explained structures. As already mentioned there are much more sites than in the two figures. The same principles apply to all the additional sites, namely that the boundaries between different Tiers become less and less.

Wigner institute data centre

The CERN Tier 0 data centre was expanded by adding an additional data centre in Budapest at the Wigner institute [62] [63]. It was a logical extension that was transparent to the end user. The two data centres are connected via two 100 Gb/s bandwidth circuits. The latency between the two data centres is around 23 ms.

4.4.2 Composition

Apart from typical Grid sites, which consist of pledged hardware from universities and research institutes, there are other types of resources that have been included into the WLCG. These are called opportunistic resources and they were introduced in order to maximise the physics throughput. They include the above mentioned HPC. The big HPC clusters can have spare resources, meaning idle cores, when for example not enough workflows are available. Another scenario would be that the available computing power is not enough for some workflows in the queue, which are then waiting until more resources become available. Significant effort has been put into the integration of HPC resources [64]. The WLCG experiments can use these idle resources, like for example at NERSC or Titan, without additional cost.

Other opportunistic resources that are used by the experiments are volunteer computing. Volunteer computing is a concept that lets PC owners donate some of their

spare computing resources to science, as an example. In ATLAS this project is called ATLAS@HOME [65]. The contribution from volunteers fluctuates, but looking at the contribution from 02.01.2018 to 02.02.2018 (PanDA monitoring), volunteer computing provided around 2486920 CPU hours for MC simulation. This corresponds to 3% of the overall MC simulation CPU consumption that took place on the whole WLCG during that period. Overall, ATLAS@HOME produces around 2% of MC simulation events [66]. In that sense, volunteer computing contributes as much to the computing as a Grid site. In January, according to the PanDA monitoring, ATLAS@HOME provided around three times as many CPU hours as GoeGrid for production jobs.

Commercial Cloud computing is already being used by several of the LHC experiments to increase their computing capacity. The CMS experiment, for example, used Amazon's AWS over a period of one month on a scale that increased the overall computing capacity of CMS by 33% [67]. In September 2015, ATLAS performed a scale test on Amazon's EC2 that successfully processed 437000 ATLAS event generation and simulation jobs [68]. These examples highlight that the experiments recognise the importance of Cloud computing and are actively working on integrating these resources.

In Figure 4.5 an overview over the overall CPU consumption, including opportunistic resources, is given. The diagram is taken from the ATLAS dashboard[7]. It is important to note, that the CPU consumption does not translate one-to-one to the physics throughput. Some machines or CPUs can be faster than others and process more workload in the same time. Of special note is that HPC (light blue and green, around 13% and Cloud computing (yellow, around 9% make up over 20% of the overall ATLAS resources. The biggest contribution of around 78% comes from the Grid.

The distribution of the computing in the WLCG and the addition of other resource types makes the whole WLCG very heterogeneous. Each computing cluster is responsible for purchasing their own hardware, therefore the WLCG consists of many different hardware components from many different vendors.

There is also no unified policy for decommissioning old hardware. This is why many generations of hardware can be found across the WLCG. The hardware therefore varies between sites as well as within a site. This makes it difficult to predict how long a given set of workflows will run on the Grid. The prediction of workflow durations is attempted in Chapter 7.

Storage

There are several possibilities to store data. The first differentiation can be made between the underlying hardware, namely tape storage, hard disk drives (HDDs) and solid state disks (SSDs).

Magnetic tapes have a long lifetime and the cost per Gigabyte (GB) is the lowest.

[7] (02.02.2018) dashb-atlas-job.cern.ch

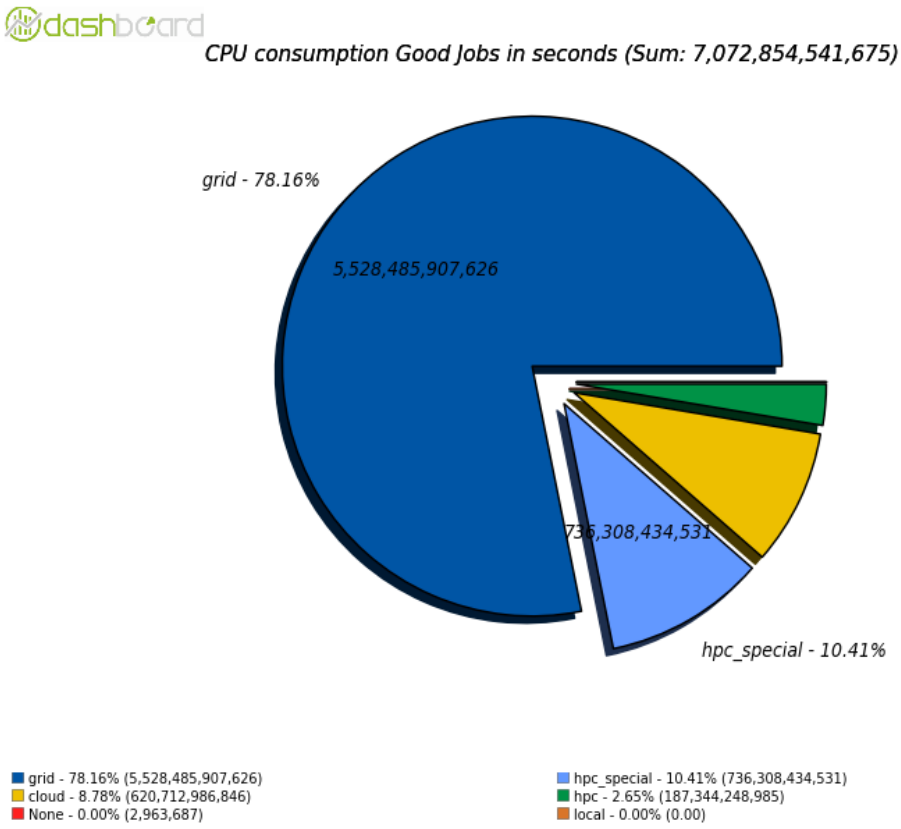

Figure 4.5: Comparison of CPU hours provided by resource type. (ATLAS dashboard)

Typically, tape cartridges are stored inside libraries. The data is read by mounting a cartridge on a tape drive. Usually there are many cartridges per drive. The read speed for sequential reads once a tape is mounted exceeds even those of disks. The downsides of tape are that random reads as well as the reading of many small files is slow. This is due to the fact that to access data at the end of a tape, the whole tape has to be mounted and then wound to that position. If all tape drives are already occupied, the read job has to wait in a queue, which increases the read time as well. In addition, it can be that the files are distributed over different cartridges that have to be mounted and unmounted in order to access them.

HDDs contain spinning magnetic disks that are always mounted, either in a storage system or attached to a computer. Therefore the possibly large overheads of the tape system are avoided. On the other hand they consume power while being idle, in contrast to a tape archive. HDDs are slightly more expensive than tape in terms of cost per GB. They have a good sequential and random read/write speed. The input output operations per second (IOPS) a hard disk can perform, are at the order of one to two hundred. A negative aspect is their relative short lifetime, which can be understood from the maximum three to five year warranty that manufacturers give.

SSDs are the most expensive storage solution. They deliver the best performance in terms of read/write speed. The IOPS an SSD can perform, are at the order of tens of thousands. Their lifetime is rather high, because they have no mechanically moving parts. Instead, the lifetime depends on how often and also how much data is written on them. SSDs incur the most cost per GB.

To get the maximum performance for a minimum in cost, all three of these storage hardware types are used, serving different use cases. Magnetic tapes are usually used for archival purposes. At CERN, physics data is archived in the CERN Advanced STORage manager (CASTOR), which is a hierarchical storage management system. In Figure 4.6[8] the extent and fast growth of data that is stored within CASTOR, reaching up to 200 PB, can be seen.

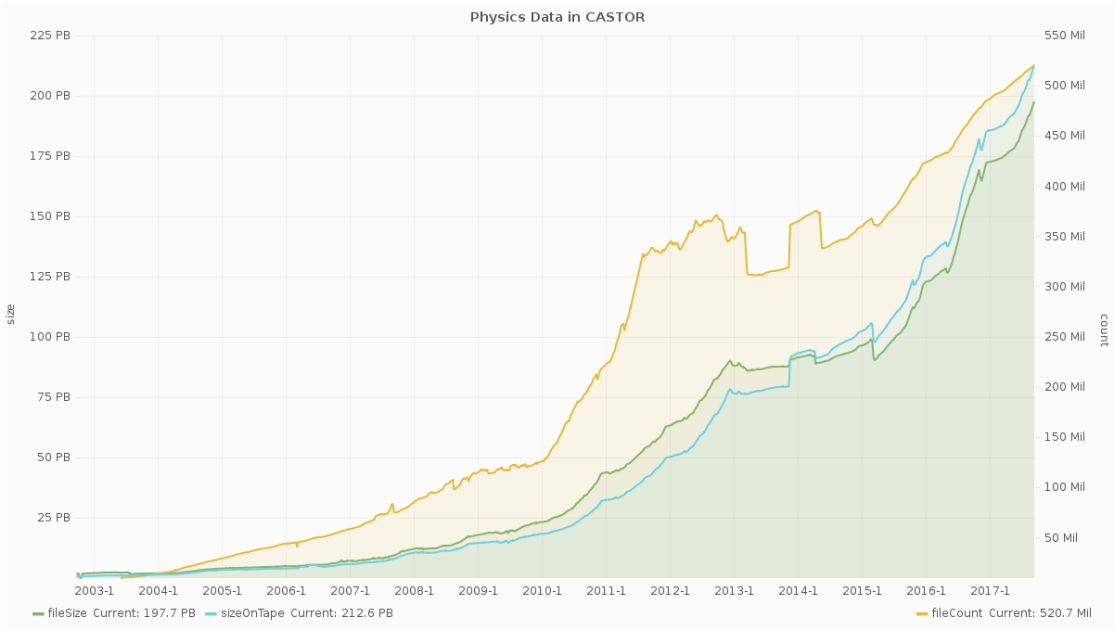

Figure 4.6: Historical development of the amount of physics data stored within CASTOR. The green and blue curve depict the file size and the size of the data on tape in PB. The yellow curve indicates the number of files stored within CASTOR.

Due to the technological evolution, tape cartridges can usually be repacked within their lifetime. After a repack they have a higher data density and can therefore store more data. Tape has a high lifetime and CERN has even stricter requirements on when to decommission a cartridge. In addition, regular checks are performed. Even though all these securities are in place, CERN lost some data due to the contamination of tapes with

[8] From the official castor webpage (http://castor.web.cern.ch/ http://castorwww.web.cern.ch/castorwww/namespace_statistics.png , 13.09.2017).

particles of concrete. In order to prevent this, a monitoring system has been installed. This illustrates that tape archival is not 100% guaranteed to preserve all data [69].

The most common use cases for HDDs are for storing frequently accessed data. At CERN, physics analysis data is stored within EOS, a disk-only storage system [70] [71]. It provides data access with low latency to physicists. Figure 4.7[9] shows the development of the EOS usage and space.

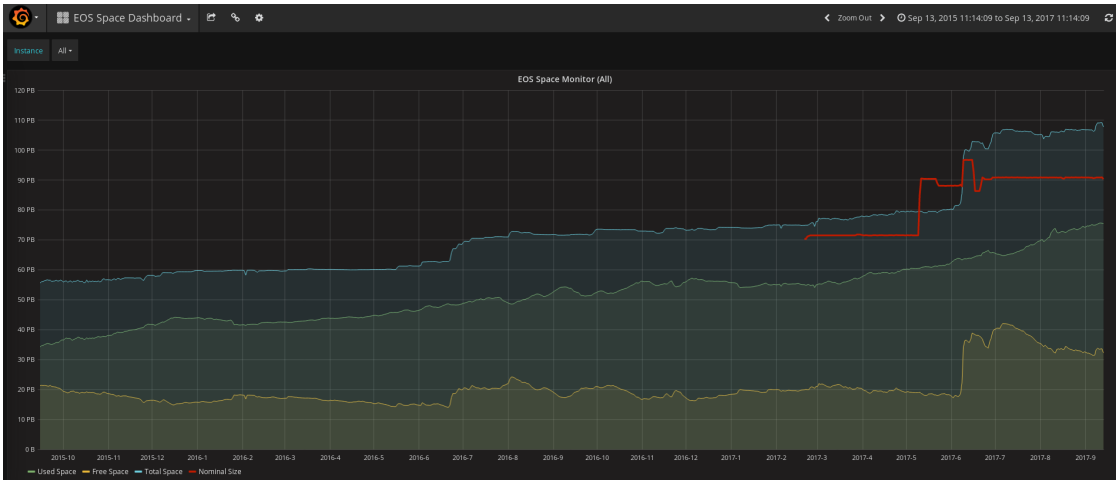

Figure 4.7: Development of the amount of physics data stored within EOS over the last two years. The green curve that depicts the used space rises steadily to about 76 PB.

Different Grid sites use also other disk storage technologies, such as e.g. the Disk Pool Manager DPM [72] or dCache [73].

For all use cases of HDDs, SSDs would be the better alternative, if it were not for the cost. Generally, SSDs are used wherever the speed of HDDs is not sufficient and would be a big bottleneck.

The ratio and area on which different storage technologies are operated may change in the future, as the prices develop differently. Figures 4.8 and 4.9 show that the annual growth and revenues of manufactured HDDs become less and less. They represent numbers that were published by IBM and indicate the trend that can be observed on the whole market. HDDs have been developed for a long time and improving them becomes increasingly expensive. SSDs (labelled as NAND) are a newer technology and have more room to be developed. Development in technology in this case goes hand-in-hand with a decrease in cost per GB.

[9] From the official EOS dashboard (https://filer-carbon.cern.ch/grafana/dashboard/db/eos-space-dashboard?refresh=5m&orgId=1&from=now-2y&to=now, 13.09.2017).

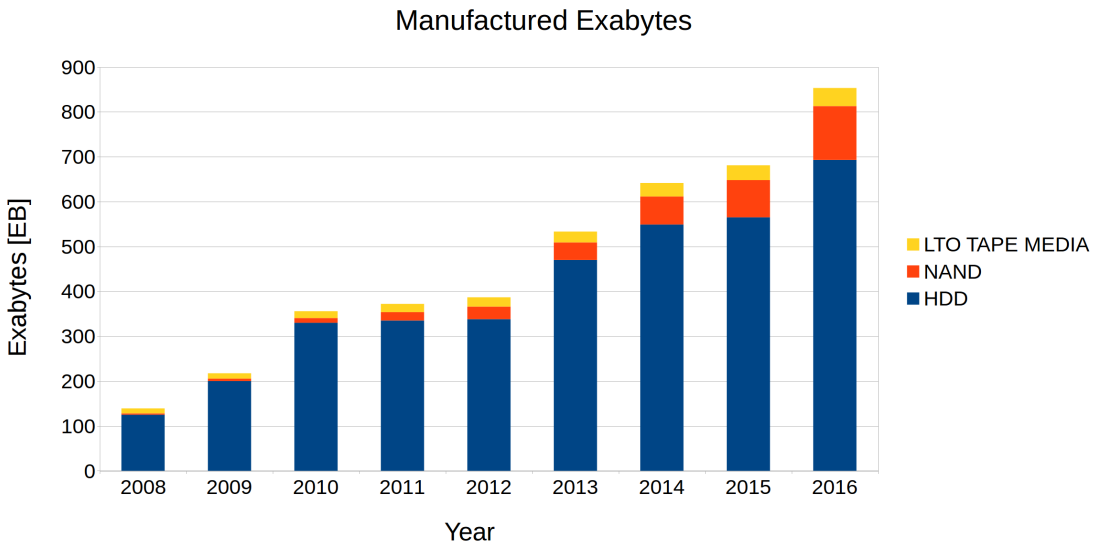

Figure 4.8: Historical view of manufactured Exabytes of HDD vs SSD (NAND). Numbers published by IBM (Decad, G and Fontana, R.).

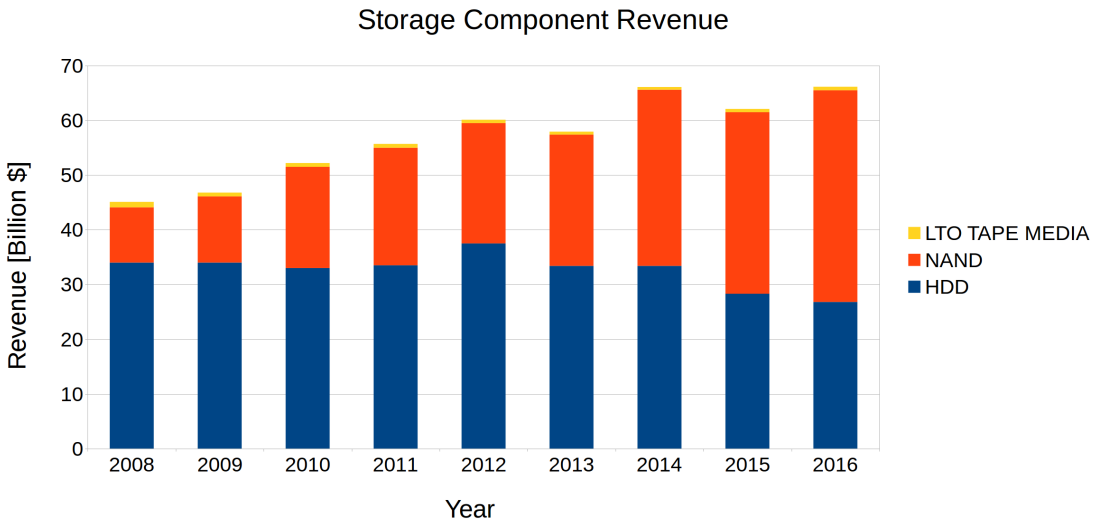

Figure 4.9: Historical development of HDD and SSD (NAND) revenue. Numbers published by IBM (Decad, G and Fontana, R.).

Looking additionally at the development of the prices for tape it can be seen that the downward trend starts to stagnate. In addition, the revenue that companies make from magnetic tapes is receding.

Technological advances are difficult to predict, but these market trends indicate that

the landscape of storage will continue to change. If crossover points for the price-per-GB are reached, it is quite possible that one technology will be replaced by another. An example would be, that instead of HDDs, only SSDs are purchased. However, it looks like this point lies far in the future.

4.4.3 Evolution

Even though most of the WLCG's infrastructure persists for several years, the WLCG is in flux. This becomes apparent when looking at the transition out of the MONARCH model. There are several possible future courses on which this development can continue.

One viable direction is to consolidate the storage further into bigger sites. None of the smaller sites would host any data anymore, except for caching. The extreme of that scenario would be to have one big storage facility per continent. The reason for this centralisation is that the storage is the most manpower and maintenance intensive part of the WLCG. Afterwards, fewer experts and personnel would be required to handle the upkeep of the storage, relieving especially smaller sites.

Another path would be to shift the computing more and more into the Cloud. This would most probably be done at the site level. Each WLCG site has pledges it has to fulfil. It does not matter where the pledged resources originate from. The resource origin could even be completely transparent from outside of the site. A second option would be to spend budget on the WLCG level for Cloud computing, for example during times when there is a peak in demand. These considerations depend on whether the Cloud infrastructure will be cheaper than acquiring and running the infrastructure by the sites/WLCG themselves. The situation can be different for each site individually, depending on factors such as the country they are located in. A summary of the viability and cost considerations of the Cloud can be found in Section 9.

4.5 ATLAS computing components

Below, some ATLAS specific Grid components/implementations are introduced. These are introduced briefly, as they are relevant for an understanding of later chapters.

4.5.1 XRootD

XRootD is a system designed to enable access to data repositories [74]. It is used in HEP for concurrent access to repositories containing multiple petabytes of data. Scalability and high performance were important design parameters that were incorporated alongside with features such as authentication [74]. Throughout this thesis, when using remote data access, it is done via XRootD.

4.5.2 Athena

Athena is the ATLAS control framework. It handles all levels of ATLAS data processing, such as simulation, reconstruction, and analysis [75] [76]. Throughout this thesis it was used to execute the different workflows that were tested and examined. Athena was created with the goal in mind to keep the data and algorithms, as well as transient and persistent data, separate.

Most notably is that Athena includes a performance and resource monitoring, which was used in some cases to compare the performance of different VMs with each other [76]. Also worth noting is that Athena uses Python as a scripting language. This allows for a job steering and configuration, that can be understood and reproduced easily. In this thesis, multiple completely different workflows are used. Each of these uses tens of thousands of different lines of code and there are several hundreds of different software versions and millions of different input files. By providing the steering and configuration files for the different workflows (in the Appendix), each of the used workflows can be reproduced.

4.5.3 AthenaMP

AthenaMP (Athena Multi Process) is the framework that lets Athena run in a multi-core environment.

ATLAS works hard on not running into the hard RAM limit. Saving memory is one reason why many workflows have been parallelised within the multi-process framework AthenaMP. The idea behind this is, that a single-process forks into multiple ones, which share parts of their memory. The shared memory reduces the overall memory footprint on a multi-core machine, by decreasing the redundancies.

An important setting that is inherent to AthenaMP is the number of parallel processes that it spawns. This is steered by setting the environment variable "ATHENA_PROC_NUMBER". Setting it to zero results in single-process execution. On the WLCG this setting is used to set the number of parallel processes equal to the number of cores that a VM on the Grid has, which can deviate from one VM to another. In Chapter 8 it is additionally used to deviate from this setting and run a larger or smaller number of parallel processes.

4.5.4 PanDA

The Production and Distributed Analysis (PanDA) workload management system is responsible for processing Monte-Carlo simulations, performing the data reprocessing and executing the user and group production jobs [77].

In Figure 4.10, it is shown how jobs and production jobs are submitted by the user and the production managers. All job information and the task queue is then handled centrally by PanDA [78]. This means PanDA takes care of the entire scheduling. Depending on the job requirements, the jobs get subsequently scheduled to matching available resources. This is done using the pilot model [79] [80] [81]. Pilot jobs are

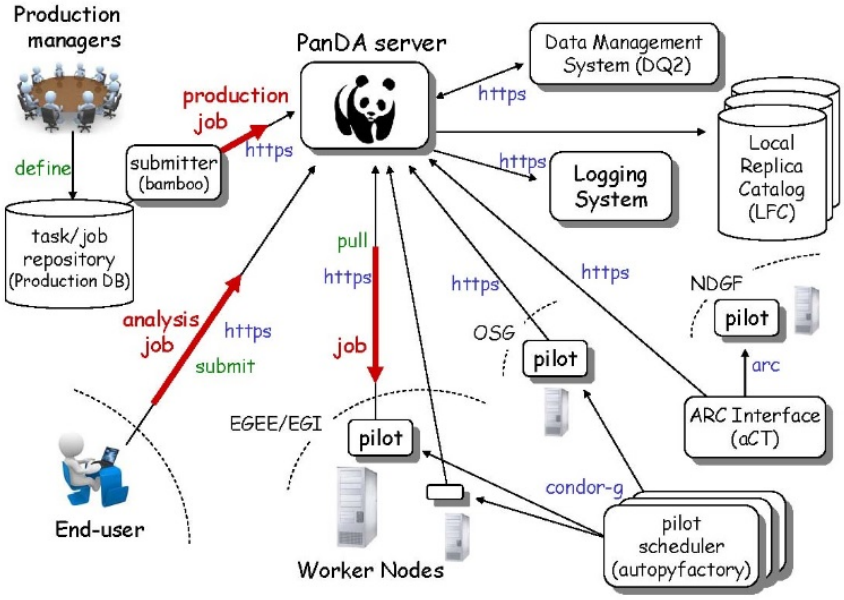

Figure 4.10: Schematic of the PanDA System [78] - DQ2 has been updated to Rucio.

basically "place-holders" for the actual payload that have been sent by the pilot factory to a batch system or Grid site. It prepares the computing element, and then pulls the job, executes it, and cleans up afterwards. This means the pilot job also handles the data stage-in and stage-out. All tests performed in this thesis were run in a controlled environment, therefore outside of the Grid, PanDA, and the pilot model. This is important to keep in mind for data staging considerations, as it is usually done by the pilot, see Subsection 5.3.2.

Another important aspect of PanDA is its monitoring capability. It collects a plethora of metrics of each job, containing for example: the CPU consumption time, the job duration, and the input size. This information can be accessed from the web and is stored and replicated to multiple places, one of which being the analytix cluster at CERN IT. The analytix cluster was used to perform data analytics in Chapter 7.

4.5.5 Rucio

Rucio is the new version of the ATLAS Distributed Data Management (DDM) system [82] [83]. It evolved from Don Quijote 2, which was previously used. It handles the accounts, the distributed storage systems and the distribution of the ATLAS data, including files and datasets. For easy usage it has a CLI and a web interface which makes the replication of datasets easier for users. Rucio was used to locate, move, and replicate input datasets for all the workflows that required input data in this thesis. It was important especially for moving the data, when investigating the workflow performance

in dependence to the input data location, see Subsection 8.2.2.

4.5.6 JEDI

The Job Execution and Definition Interface (JEDI) is a PanDA component that was implemented in order to have a workload management at the task level [84]. It translates tasks definitions from the Database Engine For Tasks (DEFT) that user requested into jobs that are then executed by PanDA.

4.5.7 CVMFS

The CernVM File System (CVMFS) is used across the HEP experiments in order to access their software and conditions data [85] [86]. It is a read-only file system capable of delivering the necessary data to all different kinds of VMs located on the WLCG via HTTP, making use of caching. It acts like a cache for the ATLAS software and data. A short investigation into the impact of the CVMFS cache on a workflow was done in Subsection 5.3.2. Throughout this thesis, all VMs were using CVMFS.

4.5.8 Tags

The TAG data is metadata, containing information about key quantities of events, which make it easier and faster to select specific events for a physics analysis [76] [87]. Individual physics events can be identified and selected via their TAG data, that is stored in a relational database.

4.5.9 AMI

The ATLAS metadata is accessed via the ATLAS metadata interface (AMI). It enables the aggregation of distributed meta data and the retrieval in web applications [88]. It is the tool for dataset selection within ATLAS [89]. The AMI components include the Tag Collector, which manages the various releases of the ATLAS software [88]. The processing history of datasets is saved using the AMI-Tags interface [88].

4.6 General concepts

In this section, further concepts are introduced, that are required for the understanding of the work done in this thesis.

4.6.1 Benchmarking

Especially in distributed computing almost all hardware and software setups differ from another. In order to have an understanding of how this difference impacts the performance, benchmarking is used. Benchmarking means executing the same defined set of operations on a defined dataset in a controlled environment.

Benchmarks are executed on different systems and the individual performances are compared to a reference. The resulting relative performance can be measured for a specific operation, such as disk read or more complex combinations as in the case of HEPSPEC06 (see Subsection 4.6.1). Ideally, a Benchmark is representative of all the processes that will actually run on the system, in order to get the most useful results.

This is especially important when trying to predict the performance of a workflow on a machine and in the context of Cloud Computing, as will be seen in Chapter 7. When wanting to purchase Cloud resources, a customer has to choose between different providers. Within each provider, better hardware costs more. In addition, the hardware performance between the providers will differ, even if the underlying hardware was the same. Reading the hardware specifications the Cloud provider publishes, does not make it possible to make an estimation of the job-performance [90]. The dilemma is, for example, whether to acquire the more expensive hardware or to acquire more of the cheaper hardware instead. This decision in view of cost-efficiency cannot be made without benchmarking.

In the end however, benchmarking alone cannot provide all the answers. This is due to the actual workflows differing from the benchmarks or from changes within the environment, such as neighbouring VMs. What benchmarking can provide, is a good first order estimation.

There is another type of benchmarking, called passive benchmarking [91]. Instead of running a dedicated piece of software that only produces a measure of the performance, a look at the actual workflows can be taken. Passive benchmarking therefore does not use additional time on the infrastructure. The results are however workflow specific and not universal. In addition, the workflow composition has to be of workflows that would all perform similarly on the same machine.

Computing power

The High Energy Physics - Standard Performance Evaluation Corporation 2006 (HEP-SPEC06) benchmark is a subset of the SPEC CPU2006 benchmark tuned for HEP [92]. It stresses the CPU with operations and algorithms that are common for the HEP community [93]. The HEP-SPEC06 benchmark is resource consuming as it takes several hours to complete [94] [91]. It is therefore usually run only once. It does not account for changes in configurations, updated software or the environment. Understanding the infrastructure is also important for accounting, here the CPU consumption is calculated in HEP-SPEC06 seconds (HS*s) - not only for ATLAS [95], but all the LHC experiments [96].

4.6.2 Storage

Over the years, the LHC data and storage strategies evolved alongside the WLCG. In ATLAS, all Grid jobs, except the raw data reconstruction at the Tier 0, are handled by a central PanDA system that manages all workflows. As a consequence, the user should only see a single computing facility. PanDA faces several challenges, the most prevalent

difficulty is to provide the CPUs with the required input data. In the current approach, this is solved by running the jobs on sites where the input data is already available. The logical consequence of this data locality is that if some data is sparsely distributed but very popular, the sites that have this data will receive a disproportionate amount of jobs, while other sites are idle.

As a fail over mechanism, data can be fetched from a remote location to the local disk. This cannot be done for every job, because there is only a limited amount of bandwidth, which would be quickly saturated, leading to idle CPUs that are waiting for their downloads to finish.

A similar third alternative is to read the input data on the fly, meaning event by event, from a remote location. The same limitations as above apply.

There are multiple mechanisms in place to mitigate these negative effects. For one, popular datasets are automatically distributed to multiple sites, so the job load is more distributed. This is handled by the Distributed Data Management (DDM) system.

Additional complexity enters when looking at the different storage media. The above cases only considered data stored on hard disks, but as described in Subsection 4.4.2 there are also magnetic tapes to consider. For jobs that have to access data that is only available on tapes, such as the reprocessing, a delay is to be expected, as the data is first copied from tape to disk. Since the amount of tape drives that can read cartridges is limited, this delay can be long. Therefore for large campaigns the experiments do an organised tape-to-disk staging before the job execution. The breakdown of tape usage can be seen in Figure 4.11.

dCache

One storage middleware system that solves some of the above mentioned difficulties is dCache [73] [97] [98]. The dCache system unifies heterogeneous disk storage systems under a single file system tree. It optimises access to tape storage, handles hot spots, load balancing, and replication.

4.6.3 Swapping

The swapping mechanism involves the fact that the CPU performs its I/O operations through the RAM [99] [100] [101]. Swap space enables the kernel to run processes that exceed the physical memory capacity. This is done by writing pages out from memory to a backing storage, known as “swapping out”. If the pages are needed, this process is reversed, known as “swapping in”.

In most cases, and throughout this thesis, the backing storage is a hard disk. Since the disk is much slower than the RAM, the access of a swapped-out page takes significantly longer than if it was in the RAM. The time the CPU is idle due to swapping operations is called swap time. In the following chapters, the swap time is indicated in equations by the variable $Swap\_Time$.

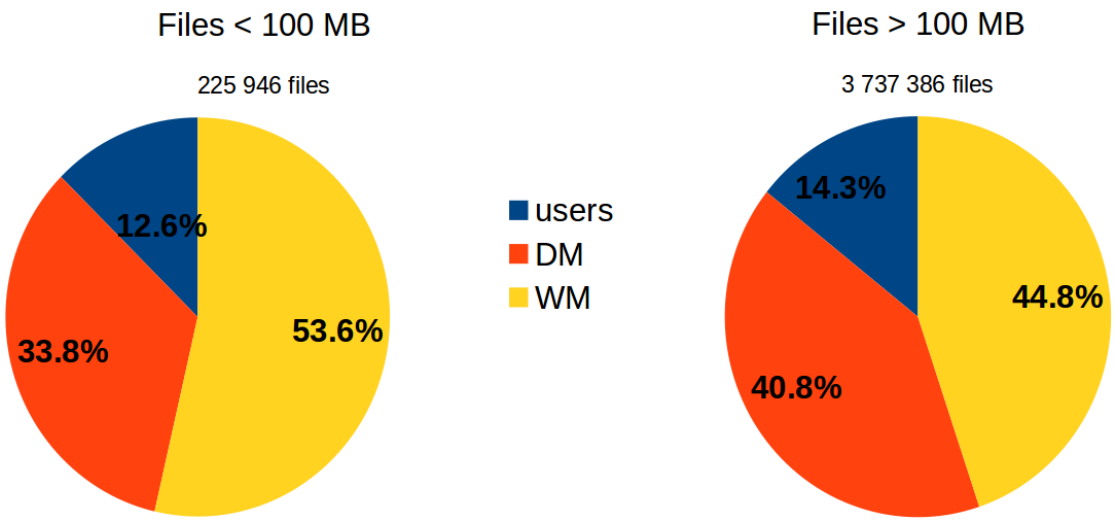

Figure 4.11: ATLAS tape recalls over six months from March to September 2017. In this case, WM refers to the on-demand staging for jobs which is composed of around 30% analysis jobs and around 70% production jobs. For DM, this means organised staging for large campaigns. Users are individuals requesting copies on disk.

If there are pages in memory that are accessed rarely or never, swapping them to disk barely impacts the performance, because they do not have to be swapped back in. This is called light swapping. Conversely, heavy swapping, or thrashing, occurs when the space in memory is not enough. Then, pages that are accessed by the CPUs more regularly are being swapped out. A vicious circle is entered and in order to swap this page back in, another regularly accessed page is swapped out. This one has to be swapped in again later, and so on. At this point the CPUs are in I/O wait longer and longer, which can slow down a workflow significantly. An exemplary profile can be found in the Appendix, see Figure A12.

4.6.4 CPU efficiency

One controversial possibility to assess the goodness of an infrastructure or workflow is to look at the CPU consumption time divided by the overall duration [102]. This metric is called CPU efficiency and a high value is considered good, because the CPU is busy. On the other hand, this favours slow CPUs, as their ratio between processing and I/O wait is larger.

4.6.5 Undercommitting

Undercommittment (UC) is the practice of using fewer resources than are available.

Even though it is wasteful to not utilise CPUs, there are scenarios in which the alternative is even worse. The prime example are applications that need more memory-per-core

than is available, which therefore have to keep one or several cores idle.

The general WLCG configuration, for example, is to have 2 GB of RAM per CPU core. In some cases, additional memory is required, for example, for the ATLAS raw data reconstruction with a high pileup, see Subsection 5.3.1. These jobs did heavy swapping with the standard machine configuration. Therefore, they took a very long time to finish. In order to shorten the workflow duration, fewer parallel processes than available CPU cores were run on the machines. This gave a better than 2-to-1 GB of RAM to CPU core ratio. With sufficient RAM to keep the swapping light, the jobs finished much faster, even though fewer CPU cores, and therefore processing power, was utilised.

4.6.6 Control groups

Linux control groups (cgroups) are used to limit the resources of processes within the same operating system [103]. Each cgroup can have different limits on the available resources such as the memory or the CPU. Processes assigned to a cgroup share the limited assigned resources.

In this thesis, cgroups are used mainly to simulate VMs with less memory. This is done by creating one cgroup with only a memory limitation and assigning all processes to it.

CHAPTER 5

Workflows

In the context of computing, there are several definitions for "workflows", for example: a workflow passes information or smaller units of work between participants within a defined set of rules, according to their dependencies and in order to reach a common goal [104] [105]. Workflows are also defined as computational tasks that are linked by data- and control-flow dependencies [106].

According to these definitions, within the WLCG (and according to ATLAS terminology, which is used in this thesis) the term workflow describes the whole system around and including a job execution. This can consist of e.g. the scheduling, the data transfers, the job execution, the monitoring and the retrying. Within the WLCG, jobs are submitted by the users and can contain any code, typically performing a data transformation or analysis. They produce outputs that are then used by subsequent jobs, whereas communication takes place through a file system.

In the previous chapters, the only workflows considered were for physics analysis. In the general picture, analysis workflows use up a not-insignificant amount of resources, but far more resources are consumed by all the processing that happens beforehand. The resource consumption distribution can be seen in Figure 5.1. The wallclock or wall time corresponds to the total duration of a workflow $Wall\_Time = T\_finish - T\_start$, where T_finish is the time the workflow finished and T_start the time it started. The name is in analogy to measuring the duration with a clock hanging on a wall.

Looking at the pie chart it becomes clear, that MC simulation, event generation and MC reconstruction require a much larger amount of resources than physics user analysis. Over 60% of the overall wallclock time is used by the combination of production jobs, compared to around 10% used by the analyses[1]. The ratio may vary amongst the different experiments, but the general trend stays the same.

[1] Tier 0 raw data reconstruction jobs are monitored separately. The monitoring data for these was unfortunately incomplete. The monitoring experts that were contacted did not provide further data.

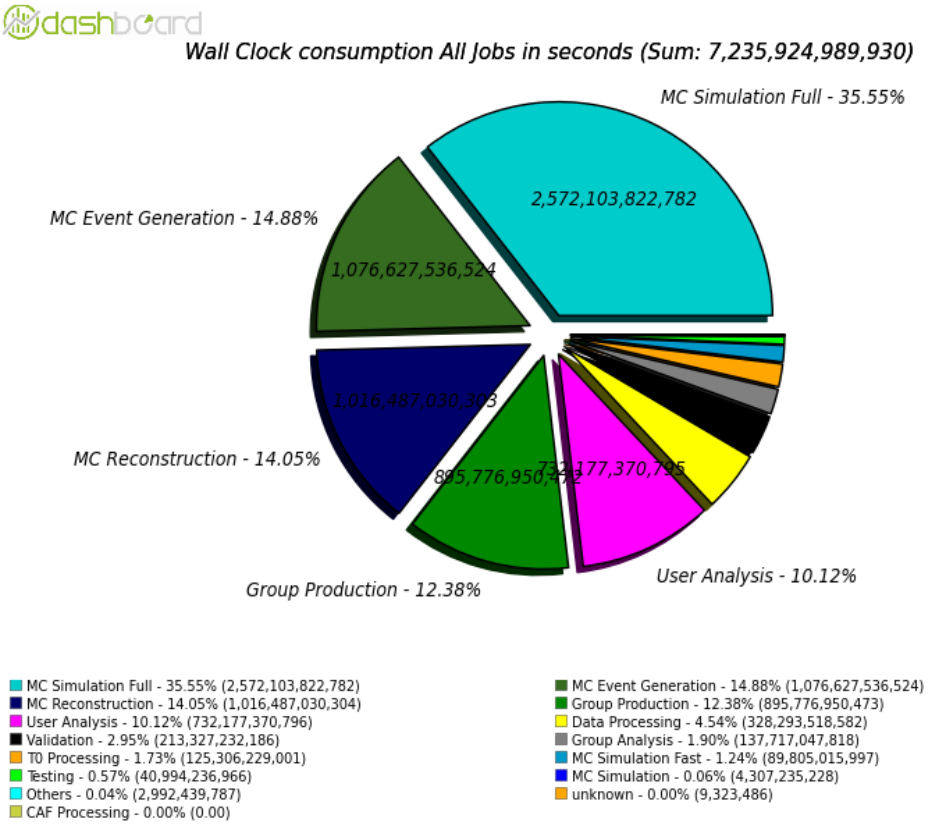

Figure 5.1: This pie-chart shows the wallclock time consumption of different types of workflows. It is a one year aggregation from 01.04.16 - 01.04.17, taken from the ATLAS job dashboard. The only caveat is that raw data reconstruction jobs are not included in the PanDA monitoring from which this plot is taken.

In this thesis, the focus lies on computing resources for the LHC. Therefore the workflows described contain the whole data processing, from the raw detector data (see Section 5.3.1) of the LHC experiments to physics analyses results.

The different workflows can be divided into two classes. This classification is done in accordance to the job model, dividing the jobs into data intensive or CPU intensive.

It is possible that an I/O intensive job also uses much CPU power. For simplification, the dominating attribute of the job is chosen as the describing class. This may simply be the most limiting factor with respect to an average infrastructure setup, also known as the bottleneck. The performance of the job is then only dependent on that component, see for example [107].

The classification has to be performed individually, by investigating i.e. what percentage of the jobs runtime is spent in CPU compared to how much is spent on I/O. The classification is done in a more abstract way, in order to have a unified classification of

the jobs. This way, CPU intensive jobs of all four experiments and outside HEP can be aggregated in the same class, even though the absolute numbers can differ by a large margin.

5.1 General model

Physics analyses are not performed on the raw recorded data, but on simulated and processed data. The whole chain of workflows can be seen schematically in Figure 5.2. The term *chain* refers to a set of linked, consecutive workflows, where the results are the input to the following workflow.

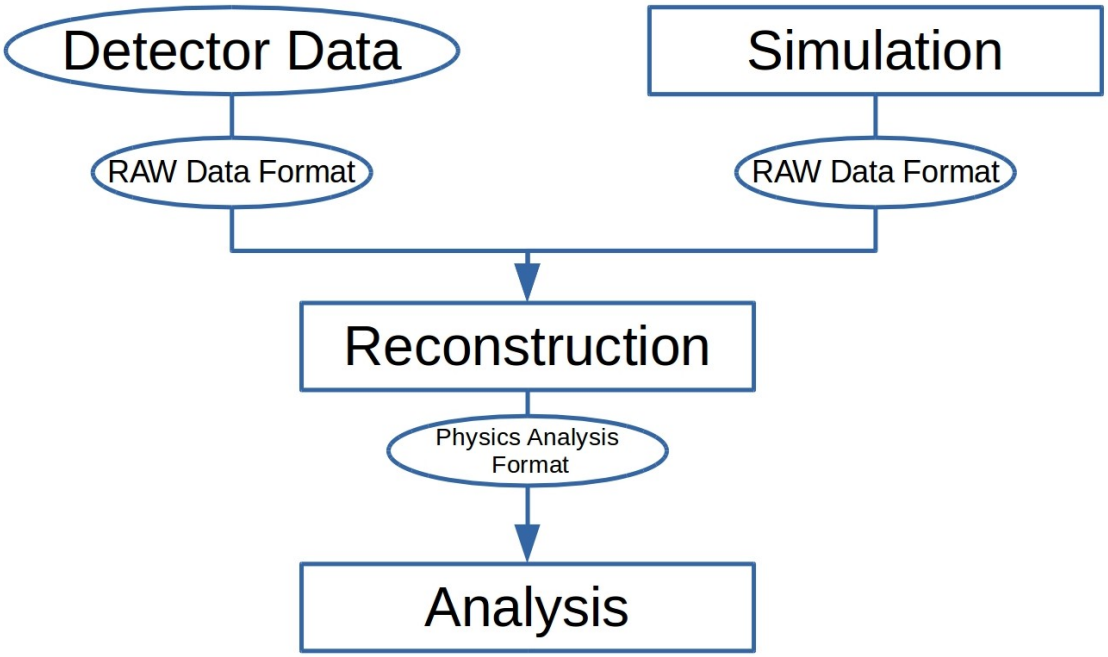

Figure 5.2: This diagram describes the chain of workflows. The detector/simulation data is computed several times until it is used for the physics analysis in the end. Workflows are represented in boxes, whereas data types are represented in ellipses.

The different experiments all perform a similar processing, although the details, such as workflows and workflow chains, differ according to their processing models. A short overview can be found in Section 5.1.1.

The general workflow consist of three job categories.

1. Reconstruction

In the reconstruction, the information of the raw detector data is combined to a higher level analysis object, meaning a different data format, that can be analysed by a physicist later on.

2. Monte Carlo simulation

   In order to confirm a physics model or to disprove it and therefore possibly discover new physics phenomena, such as a previously undiscovered particle, the real data needs to be compared to Monte Carlo (MC) simulations. These simulated proton-proton collisions and the corresponding decay products.

   In the digitisation which follows, these particles get translated to electrical detector signals, in order to stick as closely to the data processing as possible. The simulated data at this point has the same format as the real raw data.

   At the next stage, in order to treat the simulated data the same way as real data, it also undergoes reconstruction. This similar treatment is important for the interpretation of the detector data, since for the simulated signals the truth information[2] is available. After the reconstruction, the resulting data can be compared to the true input decays to make sure there are no big discrepancies. There can be small discrepancies, which have to be understood and considered in all analyses, as they will also appear in the real data. These discrepancies can, for example, appear due to a dead region in the detector, through which a real or simulated signal or particle passed undetected. This particle will not show up in the reconstructed data, even though it was there originally. Other sources of discrepancies such as software bugs, can also be found by this comparison.

3. Analysis

   The final step is the analysis of the previously reconstructed data, including both simulated and real data.

Reprocessing of the raw data is the same as re-doing reconstruction, so it is not a category of its own. A reprocessing campaign is started, if there are significant developments in either the calibration and alignment and/or the software, so that the resulting data are improved.

The data that have been processed get a tag, that is a short event summary which is used for event identification and selection. Tag data is stored in a relational database.

The previous three job categories, reconstruction, MC simulation and analysis, can be split further into several intermediate procedures. A more detailed description of these is shown in the following subsections.

[2] The truth information is information about which decays were originally simulated.

5.1.1 All experiments

Even though the detector layout and the software frameworks of the four major experiments of the LHC differ from each other, they face similar challenges and have many commonalities. Therefore, from a rough point of view the whole data processing is rather similar. The experiments all collect signals from a detector with a certain geometry. All four of them have the goal of physics analysis in the end, so Monte-Carlo simulations are required. The biggest accordance is the computing infrastructure, the WLCG. In order to give a good overview, the biggest differences between the computing approaches of the experiments are presented.

CMS

CMS has the Any Data, Anytime, Anywhere (AAA) data federation, which is helping to solve the problem of data locality. With AAA, the CMS input data can be read directly over the WAN by the jobs.

Traditional pileup mixing has to be done anew for every simulated signal event. CMS does pileup premixing. 'Premixed' pileup means that the mixing is done only once beforehand and the response is saved to pileup events in a premixed pileup dataset, which is only around 1 MB per event, compared to around 100 MB of traditional pileup input per event. This dataset is basically a minimum-bias-only dataset that is overlaid over the simulated collisions. This reduces the I/O of the job significantly.

ALICE

The ALICE experiment has much larger and more complex events with respect to the other LHC experiments. The size ranges up to 6 MB per raw data event for lead-lead collisions [96]. In order to increase their efficiency for individual analyses, the ALICE collaboration combines analyses using the same/similar datasets into one job. This is done in order to read the data only once and then process it many times, according to the different jobs. The analysis jobs are combined in LEGO (Lightweight Environment for Grid Operators) trains [108].

LHCb

One big difference in workflows for LHCb compared to the other experiments is, that a MC simulation job encompasses event generation, simulation and reconstruction. One job processes the data from the event generation to a physics analysis ready output file. Some simulation jobs even reduce the output data by filtering uninteresting events at the end.

Another fundamental difference between LHCb and, for example, ATLAS is in the pileup. It is around 1.4 collisions per event for LHCb and therefore significantly lower than for ATLAS, where it is at around 20. The LHCb workflows are adapted to this, by simulating the low pileup directly for each collision. In contrast to that, in ATLAS

the MC simulation does not include the pileup generation. The minimum bias is instead overlaid/added afterwards.

5.1.2 ATLAS

In ATLAS terminology a job can consist of several *transformations*, see Figure 5.3. Each transformation delivers, as the name suggests, processed or transformed data. These transformations can be grouped together into jobs, in order to increase the efficiency by reducing overheads. This can be seen easily, when looking into a VM where the input data for a consecutive transformation is already locally available from the previous transformation, instead of having to be staged out and in to the local disk again.

Different jobs can partly consist of the same transformations. For example the RAW-toESD transformation is contained in both a raw data reconstruction job, as well as a digitisation+reconstruction job for simulated data. The job composition is different, but some parts, some transformations, can be similar.

The diagram in Figure 5.3 depicts that all jobs within a task are comprised of the same transformations, whereas jobs of a different task can consist of different transformations.

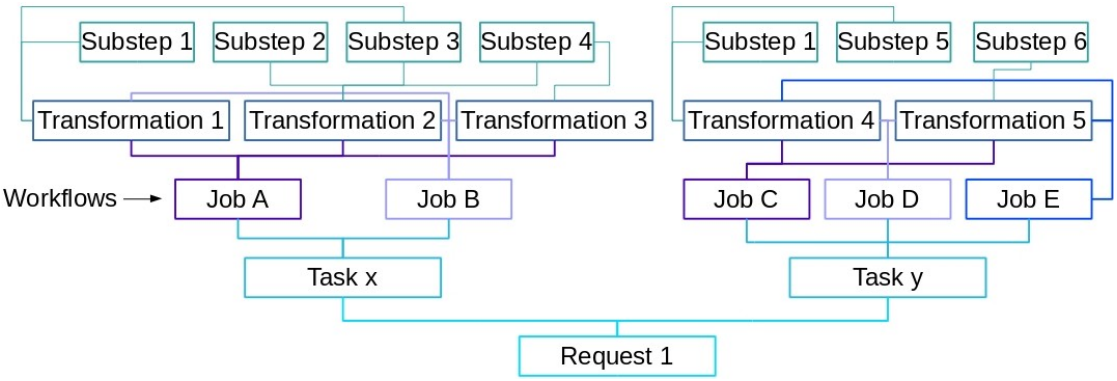

Figure 5.3: This diagram describes the general ATLAS workflow and terminology of computing.

When looking into what happens inside individual transformations, some repeating patterns can be observed. This can be between jobs inside different tasks or even requests. For example, this can include the input/output file validations that happen at the beginning/end of many transformations. Another similarity may be the fetching of conditions data that occurs in some transformations. The different parts of a transformation are called substeps, see Figure 5.3. The stage-in and stage-out are also categorised as substeps. Not many transformations within a job have similar substeps.

Below the job level are tasks. Tasks are typically comprised of many jobs that compute the same transformations, do the same thing, only on different datasets.

Due to various limitations[3], a job typically computes between 100 and 10000 events, depending on the jobtype. Therefore, in order to compute all events delivered by the ATLAS detector, there have to be many jobs doing the same processing in parallel - to finish within a reasonable time. Jobs doing the same processing on different input data are called "similar jobs". This is not to be confused with executing a job twice with the same input data, which would be called the "same job". Jobs that do not consist of the *same* transformations/substeps are "different jobs" or of a "different jobtype". This is the case even if the input dataset is the same.

On the bottom-most layer are the requests, see Figure 5.3. Requests consist of many different tasks, that do different types of processing. They are typically logically linked together, meaning that the outputs of jobs from one task are inputs of jobs of another task of the same request. Thereby requests can represent a whole processing chain, excluding analysis.

At certain points in time, data reprocessing campaigns, see Subsection 5.3.6, are started. A campaign includes all jobs, tasks or requests that were executed on the data that had to be reprocessed.

Conditions data

During data taking it is important to know the exact conditions, the state of the whole experiment, in order to achieve a precise simulation and reconstruction. This conditions data set consists of hundreds of parameters, including, for example, alignment, beam position, magnetic field map, cabling, calibration, corrections, detector status, noise, pulse shapes, timing and dead channels [109]. Each subsystem provides these parameters, which are stored in the conditions database. They are updated on the fly, some much more frequently than others. An example would be the cabling, that stays the same much longer than the alignment parameters.

5.2 Monte Carlo simulation

Monte Carlo simulation generally consists of three subsequent computations, event generation, simulation and digitisation. Each part is described in the following subsections.

5.2.1 Event generation

Mostly external, meaning non-ATLAS specific, tools are used to generate events. In the event generation, collisions (events) and immediate decay products are generated. This means everything that would happen in an actual collision up to the point when the

[3] Limitations are e.g. the maximum lifetime of a Grid job. Other considerations why the wall time should not be too long are errors. If a long job fails towards the end, much computing time is wasted.

particles hit the detector. Depending on the lifetime of the resulting particles ($c * \tau <$ 10 mm), they are already decayed by the event generator and only the resulting particles are considered to interact with the detector. Particles with a higher lifetime are then handled later on, by the simulation. Before the events are passed to the simulation, the specific beam conditions are applied.

The resulting data can be uniquely identified via the software version and inputs, e.g. job parameters, random seed [110]. For reproducibility, the processor chip architecture has to be taken into account, since it influences the pseudo-random numbers which may be generated differently, especially between different manufacturers. The information of the parents of unstable particles is preserved.

In Figure 5.4 the profile of a single core event generation job, that was run on a four core VM, can be seen. Since the job does not process any data and the output file is very small, there is very little disk activity. The job is entirely CPU bound (25% CPU usage corresponds to one CPU core), using very little memory and almost no network bandwidth. There is almost no variation of the resource usage during the whole processing. The generator that was used was Pythia v8.8186.

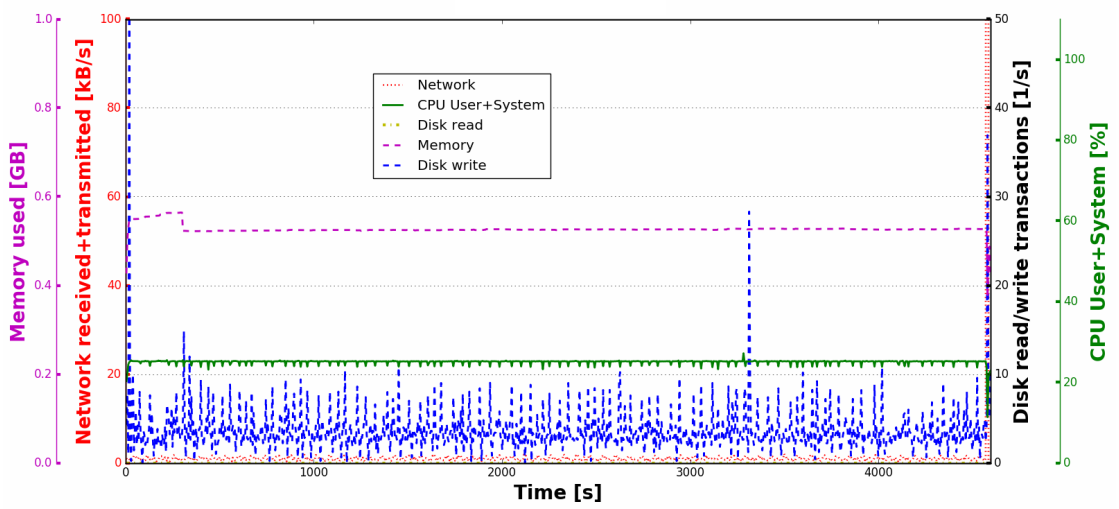

Figure 5.4: Profile of a singlecore event generation job run on the VM at CERN, processing 1000 events, using Athena version 19.2.3.6.

Event generation fluctuation

The fluctuations of the wall time of the different workflows will become extremely important for the model later on in Chapter 6. Initially, the *same* event generation job, with the same random seed input, was run 50 times on two different machines. The specifications can be found in the appendix, for the VM at CERN see Subsection A.7.2, for the VM at Göttingen see Subsection A.7.1.

In Table 5.1 it can be seen that the job duration does not fluctuate much. Though,

	Average wall time [s]	Standard deviation %
CERN VM	4488	1.59
Göttingen VM	5475	0.52

Table 5.1: Summary of executing the *same* event generation job 50 times on two different VMs, located at CERN and Göttingen.

the fluctuation is slightly higher on the CERN VM than on the one in Göttingen. The standard deviation of the sample (s) is obtained by:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{5.1}$$

where n is the number of measurements, x_i the observed values and $\bar{x}$ is the mean value of the measurements.

The standard error of the mean ($\sigma_{\bar{x}}$) is obtained by:

$$\sigma_{\bar{x}} = \frac{s}{\sqrt{n}} \tag{5.2}$$

where s is taken from Equation 5.1 and n is the number of measurements.

A plot depicting the wall time average over 1, 2 ... 50 jobs with the corresponding estimation of the standard error of the mean has been created, see Figure 5.5. It can be described as a sliding window analysis, where the step-size is one and the window-size increases by one with each step.

In order to exclude effects that appear due to the ordering of the jobs, the same plot has been created multiple times. The only difference is that the ordering of the input data points has been rearranged pseudo-randomly by hand, see Figures A1, A2 and A3 in the Appendix.

The fluctuations are very small, considering that the y-axis does not start at zero. Approaching a high number of jobs ($n > 25$) the wall time average converges within a reasonably small standard error of the mean.

The same investigation has been performed for *similar jobs*[4]. Since *similar jobs* are different from each other, it is expected that the variation in wall time will become larger than before, which is confirmed by the numbers in Table 5.2.

Figure 5.6 shows the wall time average for an increasing number of jobs.

The plot shows the behaviour for the jobs at Göttingen. The same plot showing the results for the CERN VM can be found in the Appendix, see Figure A4. The fluctuations of the wall time average increased together with the standard deviation. Overall these

[4]Similar jobs have different random seeds as inputs for each job

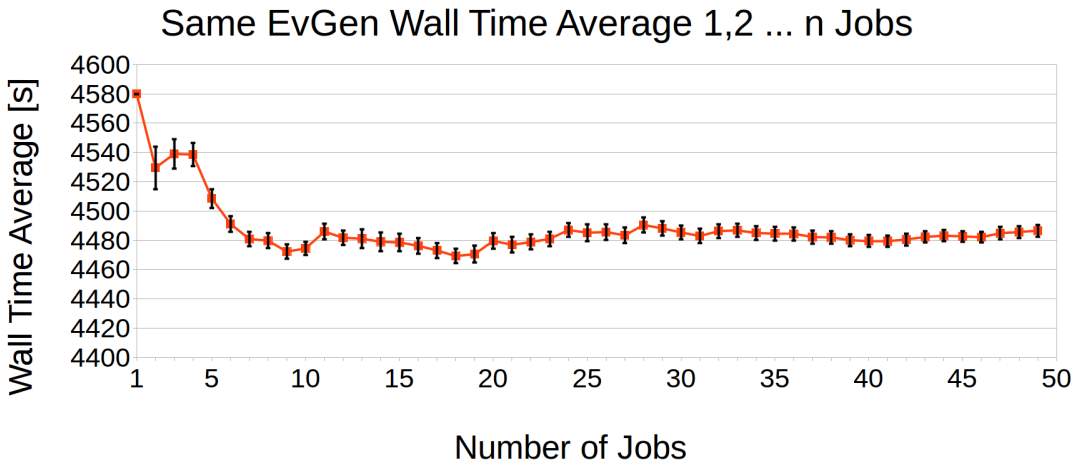

Figure 5.5: The wall time average over an incrementing number of the *same* event generation jobs, run at Göttingen, starting at one. The black error bars show the standard error of the mean (see Equation 5.2). Note: in order to improve readability, the y-axis does not start at zero.

	Average wall time [s]	Standard deviation %
CERN VM	4510	3.97
Göttingen VM	5368	3.34

Table 5.2: Summary of executing 41 *similar* event generation jobs.

fluctuations are still very small and they also converge - albeit after a larger amount of jobs are included.

Changing the workflow to an older version, which was used in 2015, does not show different results in terms of fluctuations.

5.2.2 Simulation

The simulation workflow simulates the detector and physics interactions of the particles with the detector. This is done by the GEANT4 [111] toolkit which models the physics and particle transportation through the detector. It takes the resulting data from the event generation as input.

The truth information from the event generation is kept and particles from the simulation step are added.

In Figure 5.7 the profile of a MC simulation is shown. The whole processing is CPU bound with very little disk and network activity. In this case the job ran on four cores in parallel. There is little variation of the resource usage during the processing, from about

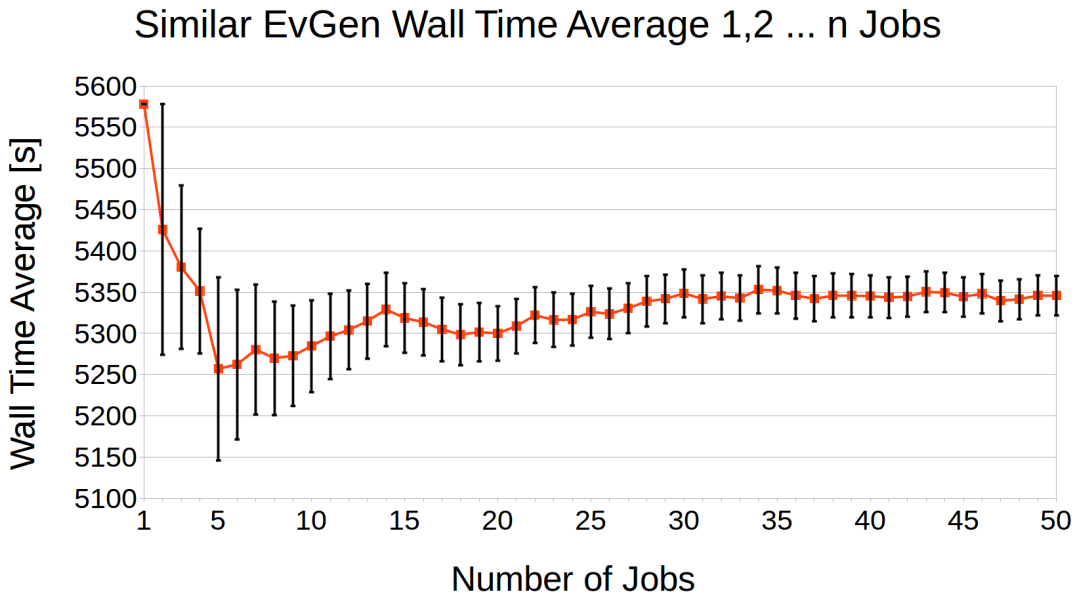

Figure 5.6: The wall time average over an incrementing number of *similar* event generation jobs, starting at one, at the VM at Göttingen. The black error bars show the standard error of the mean (see Equation 5.2). Note: in order to improve readability, the y-axis does not start at zero.

200s to about 5600s. Towards the end of the job, after 5500s, it can be observed how the individual processes finish the simulation of the last events at different times and the CPU usage decreases in steps (100% to 75% to 50% to 25%) to zero. The memory profile of the job is rather flat and a low RAM requirement can be seen.

Monte-Carlo simulation fluctuation

The fluctuations of the wall time of the different workflows will become extremely important for the model later on in Chapter 6. The variation amongst repeating the *same* simulation job were much lower (0.78% for CERN and 0.28% for Göttingen) than for *similar* simulation jobs. This is why only the results for *similar* jobs are shown, see Table 5.3, which in any case includes the *same* job fluctuations.

Figure 5.8 shows the wall time average over an increasing number of *similar* Monte-Carlo simulation jobs. The outputs of the *similar* event generation jobs in Subsection 5.2.1 were taken as inputs for these simulation jobs.

The plot depicts the results for the VM in Göttingen. The same plot, showing the results of the VM at CERN can be found in the Appendix, see Figure A5. Overall the fluctuations are rather small, in the order of a few percent, compared to the wall time. Towards a higher number of jobs ($n > 30$) a convergence can be seen.

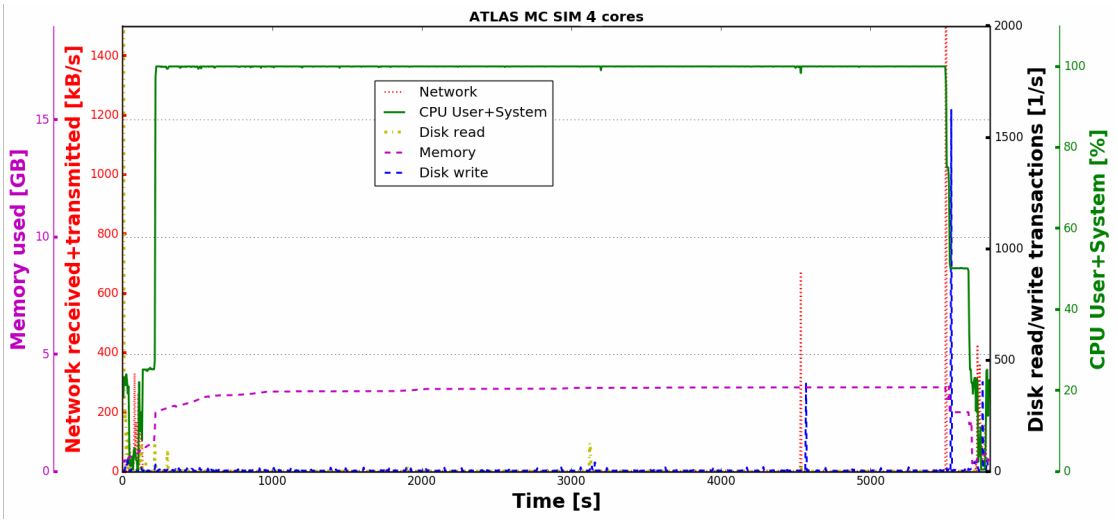

Figure 5.7: Profile of a multicore simulation job run at a 4-core VM at CERN, processing 100 events, using Athena version 19.2.3.3.

	Average wall time [s]	Standard deviation %
CERN VM	5010	4.37
Göttingen VM	3097	4.42

Table 5.3: Summary of executing 26 *similar* Monte-Carlo simulation jobs at CERN and 38 *similar* Monte-Carlo simulation jobs at Göttingen.

5.3 Reconstruction

A reconstruction workflow takes detector-hits data and transforms it into physics objects that can be used by a physics analysis. This is done by deriving particle information from the signals of the detector components. By using the geometry of the detector, individual particle tracks in the detector can be calculated. The track then provides information that can identify the individual particle properties and its identity, see Chapter 3 about the ATLAS detector. All of this is done using multiple, sub-detector dependent, algorithms, which are in the end combined. Combined information is for example jets or missing transverse energy.

The same reconstruction is applied to the raw data as well as the simulated data. The only difference is that simulated data usually has to include the digitisation step, which happens right before the reconstruction.

In addition, for simulated data the truth information is available, which can be used to validate and improve on the reconstruction.

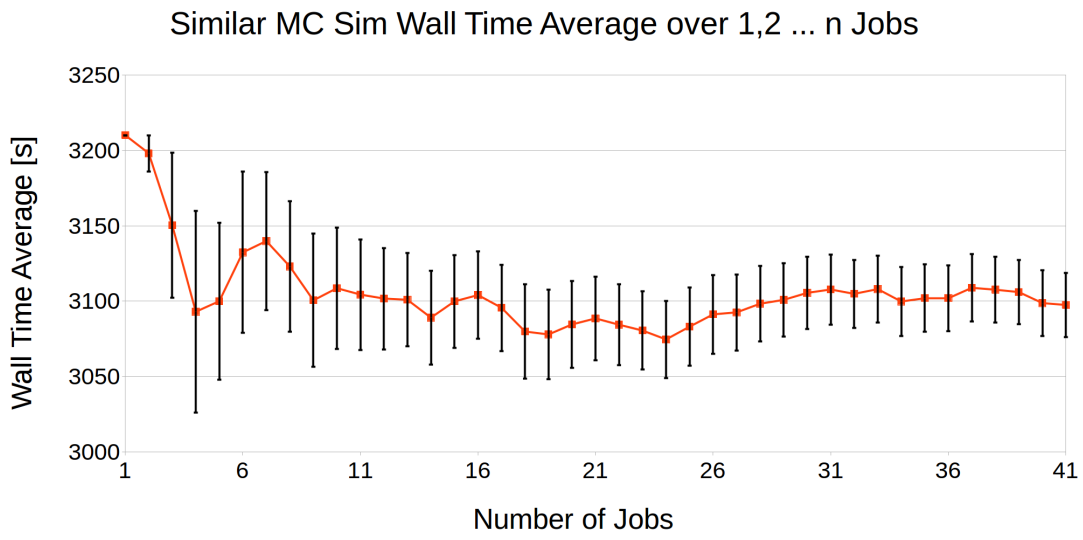

Figure 5.8: The wall time average over an incrementing number of *similar* Monte-Carlo simulation jobs, starting at one, at the VM at Göttingen. The black error bars show the standard error of the mean (see Equation 5.2). Note: in order to improve readability, the y-axis does not start at zero.

5.3.1 Raw data reconstruction

The individual transformations that the data undergoes are RAWtoESD, ESDtoAOD, ESDtoDPD and POOLMerge.

5.3.2 Raw data reconstruction profile

In the previous subsection, individual processing steps were introduced. These can also be seen when looking at the job profile, as they have different requirements from the hardware.

In Figure 5.9, the profile of a raw data reconstruction job is shown. The first aspect to look at is the green solid line that highlights the overall CPU usage. On a four core machine, 25% CPU usage corresponds to one core being used. There are three different states that can be observed, namely zero, one, or four CPU cores that are used.

Secondly, the dashed purple line shows the memory usage of the job. It roughly follows the CPU usage.

The finely dashed red line depicts the network activity. Here, notably at the very beginning a high activity can be observed, depicting the stage-in of the input data from a remote storage to the local disk. Throughout the rest of the job the network activity

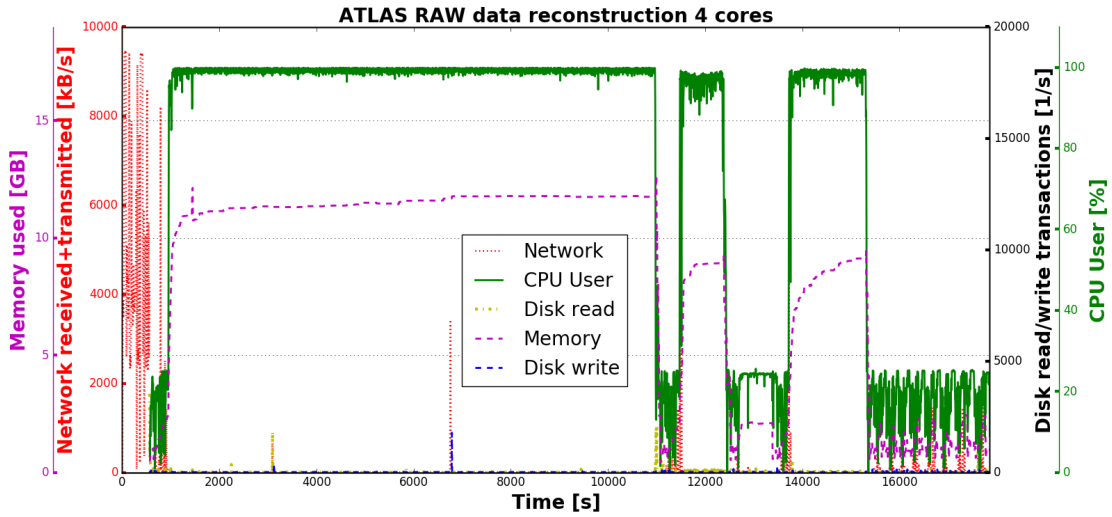

Figure 5.9: The profile of a raw data reconstruction job, run with AthenaMP on a machine with four CPU cores.

remains low.

Finally the disk activity, depicted by a blue dashed line (disk write) and a yellow dashed/finely-dashed line, can be seen. Overall the disk activity is rather low throughout the job, even though the input data is read from disk and the output written back to it.

The overall profile corresponds to the different transformations described in Subsection 5.3.1. In the beginning, up until about 500 s, a high network activity can be seen. This corresponds to the stage-in of the data. Afterwards, from about 500 s to 11200 s, the RAWtoESD processing takes place. The ESD data is then processed in the ESDtoAOD step from about 11200 s to 12500 s, followed by a merging step that lasts until 13700 s. In the end, a second processing of the ESD data from ESDtoDPD happens from about 13700 s to 15300 s, which is again followed by a merging step that lasts to the end. The merging steps are serialised/synchronised, meaning during merging only one CPU core is used.

Technically creating DPDs is part of the group production, see Subsection 5.4.1. However, it turned out that in some cases this approach can be inefficient if processing is duplicated across the different groups. Therefore a common set of primary DPDs is centrally produced during reconstruction and the groups can then use it further.

Network usage

Reconstruction is a workflow that has a large amount of input data. In this example case, it was ~ 2.6 GB per job. These 2.6 GB correspond to 3025 events, resulting in

an average event size of ~ 860 kB. The workflow standard is to stage the data to disk before the processing. This is not different from any other download and therefore does not require a special examination. In the following, the remote reading is investigated.

In order to distinguish between the noise and the network traffic of interest, a baseline was taken. The baseline of the Network traffic that was captured was very low and almost steady. The background peaks up to 2 kB/s. This is negligible compared to the traffic that was observed from the job, where the smallest peaks of interest start at ~ 400 kB/s. For these reasons, the background is ignored in the following plots.

The following Figure 5.10 depicts the network profile when reconstructing 100 events, using remote input data. The data was captured via the tcpdump command and is displayed in wireshark[5]. The graph was cut off at around 1450 s in the horizontal direction for better readability. Originally it simply continued with the small bumps until ~ 2550 s, when the job ends. Furthermore, the CVMFS cache was already filled.

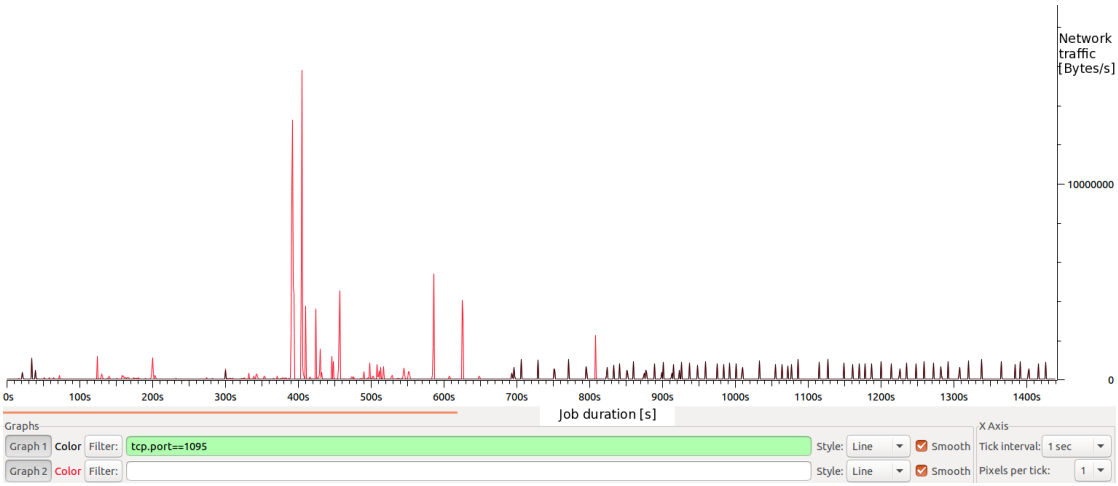

Figure 5.10: Network profile of a job that reconstructed one hundred events. It was cut off at approximately t = 1450 s for better readability.

The red curve depicts all traffic, whereas the black curve is what remains after applying the filter that restricts the ports to port 1095, used by xroot.

Around 53 little black and red peaks can be observed. In the plot without the cut-off, the total amount of little peaks is ~ 100, indicating that the individual events are read/streamed one after the other as input during job execution. Due to the filter, it is possible to easily identify the source of these peaks to originate from an EOS disk server at CERN. This corroborates that they are the physics events being read. The remaining traffic, displayed by the red curve that is not overlaid by the black curve, can be seen in the conversations list in Figure 5.11.

[5]Wireshark is a tool that is used to analyse captured network traffic.

IPv4 Endpoints

Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	City
lalalalala.cern.ch	85 036	211 875 194	35 986	5 490 294	49 050	206 384 900	Geneva, 07
p05153065463047.cern.ch	20 139	84 722 649	12 261	84 168 957	7 878	553 692	Cern, 07
caproxylbp.cern.ch	12 067	33 321 162	8 888	32 850 896	3 179	470 266	Cern, 07
caproxylbp.cern.ch	8 597	27 444 433	6 214	26 996 333	2 383	448 100	Cern, 07
caproxylbp.cern.ch	7 682	16 431 436	4 724	15 533 004	2 958	898 432	Cern, 07
caproxylbp.cern.ch	4 054	13 852 038	2 700	13 560 884	1 354	291 154	Geneva, 07
caproxylbp.cern.ch	3 791	10 442 088	2 497	10 115 404	1 294	326 684	Cern, 07
caproxylbp.cern.ch	4 243	9 936 515	2 576	9 504 158	1 667	432 357	Geneva, 07
caproxylbp.cern.ch	3 815	5 611 457	2 372	5 156 199	1 443	455 258	Cern, 07
caproxylbp.cern.ch	1 950	4 330 310	1 196	4 168 778	754	161 532	Geneva, 07
caproxylbp.cern.ch	2 291	2 666 998	1 233	2 323 226	1 058	343 772	Cern, 07
p05153065513165.cern.ch	806	1 563 114	441	1 526 463	365	36 651	Cern, 07
cert-op-004.cern.ch	12 360	1 112 712	3 537	360 774	8 823	751 938	Cern, 07
l513-c-rbrmx-1-ci122.cern	3 470	242 900	3 470	242 900	0	0	-
vrrp.mcast.net	3 470	242 900	0	0	3 470	242 900	-
ghgfgfgfhfgh.cern.ch	2 230	208 972	0	0	2 230	208 972	Cern, 07
eosatlas.cern.ch	229	98 804	98	45 272	131	53 532	Cern, 07
ip-dns-1.cern.ch	406	62 803	203	46 817	203	15 986	Geneva, 07
ccwbvip14.in2p3.fr	39	27 069	19	20 121	20	6 948	Villeurbanne, B9
caproxylbp.cern.ch	128	23 064	18	1 332	110	21 732	Cern, 07
pcitgt1012.cern.ch	105	12 674	66	5 652	39	7 022	Geneva, 07
224.0.0.251	64	4 574	0	0	64	4 574	-
jenkins-master-5e27bd86	8	1 998	8	1 998	0	0	Geneva, 07
f513-c-ip122-lhp6m-10.cer	28	1 288	28	1 288	0	0	Geneva, 07
all-systems.mcast.net	28	1 288	0	0	28	1 288	-
deis-03.cern.ch	28	1 288	28	1 288	0	0	Cern, 07
224.0.0.252	28	1 288	0	0	28	1 288	-
ntp1.as34288.net	9	810	0	0	9	810	-
ns2.pmodwrc.ch	9	810	0	0	9	810	-
91.240.0.5	9	810	0	0	9	810	-
ip-time-2.cern.ch	8	720	4	360	4	360	Geneva, 07
ds1789963.dedicated.solı	7	630	0	0	7	630	Zuchwil, 18
ip-time-1.cern.ch	6	540	3	270	3	270	Geneva, 07
213.184.127.45	1	62	1	62	0	0	-
lcellall.cern.ch	1	62	0	0	1	62	Cern, 07
pinger-j2.ant.isi.edu	1	46	1	46	0	0	-
zcc3xbgwcth.cern.ch	1	46	0	0	1	46	Geneva, 07

Figure 5.11: List of all network conversations of the VM called 'lalalalala.cern.ch', where the jobs were run on. The captured traffic is from the beginning of the job to the end. This list is sorted by bytes.

The first item on the list is the machine itself. It shows the total of sent/received traffic. The second item (p051....cern.ch) is the EOS disk server eosatlas.cern.ch, which is contacted directly. The traffic of 84.7 MB divided by 100 events corresponds roughly to the expected event size of 860 kB/event. It represents the transfer of the input data. The following element, caproxylbp.cern.ch, can be separated into two things, namely CVMFS and ATLAS frontier. The Frontier caching system delivers the detector calibrations, geometries and constants that are required for the reconstruction. It also acts as a Squid cache for CVMFS. Since the local CVMFS cache was already filled, the CVMFS contribution is small. At the beginning of the job, ccwbvip14.in2p3.fr is contacted, which is the ATLAS Metadata Interface (AMI). The domain name systems can be found under ip-dns-*.cern.ch. Everything else contributes little to the traffic and can be attributed to background.

In addition, mylcgvoms2.cern.ch is contacted in order to create a proxy.

CVMFS cache

In the beginning it was mentioned that all these jobs were executed with a filled/hot local CVMFS cache.

To have an idea how an empty CVMFS cache impacts the network usage, the *same* job as above was executed with an empty CVMFS cache. The local cache was cleared as much as possible, using "sudo cvmfs_config wipecache". A quick check afterwards, using "df", reveals that the cache is mostly, but not entirely, empty, see Table 5.4.

Filesystem	1K-blocks	**Used**	Available	Use%	Mounted on
cvmfs2	4096000	**125175**	3970826	4%	/cvmfs/atlas.cern.ch

Table 5.4: Output of the 'df' command on the VM after the local cache was cleared.

Figure 5.12 depicts the changed profile.

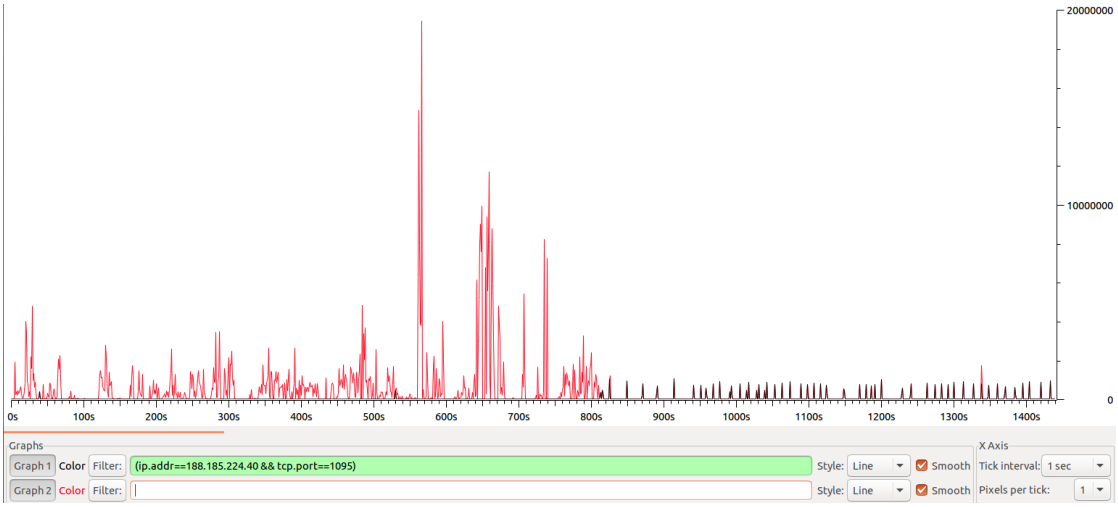

Figure 5.12: Network profile of a job that reconstructs one hundred events with an empty CVMFS cache beforehand. It was cut off at approximately t = 1450 s for better readability.

As expected, compared to the traffic with a hot cache, an increase in network traffic at the beginning of the job can be seen. Looking at the size of the cache afterwards, using "df", reveals an increase in size of ~ 1.2 GB, see Table 5.5.

RAM requirement

An important next step was to look at how the memory influences the wall time. An understanding of the RAM requirements will be needed when overcommitting, see Chapter

Filesystem	1K-blocks	**Used**	Available	Use%	Mounted on
cvmfs2	4096000	**1317537**	2778464	33%	/cvmfs/atlas.cern.ch

Table 5.5: Output of the 'df' command on the VM after a reconstruction job was run (cache filled).

8. This was performed by putting a limit in place, that restricts the amount of available memory on the VM. This limit was then varied while monitoring the workflow performance. Initially the RAM limitation was achieved by using cgroups, see Subsection 4.6.6. An even easier solution was to allocate memory to another process instead (see Appendix 1). This easier approach was adopted later on. One encountered limitation was that when lowering the available RAM too much, at some point the VM runs out of swap space. Then the Linux kernel Out-Of-Memory Killer [112] starts killing processes. This is a protective measure of the kernel in order to stay operational. After this happened, the swap space was increased. In general, VMs performing ATLAS workflows have a limited swap space, the maximum size is twice the amount of available RAM.

In Figure 5.13, the wall time, as a function of the available memory is plotted. The interesting part is that the jobs at 6 GB of available memory use over 4.5 GB of swap space, but a significant slow down can only be observed below 4.5 GB of memory. Indeed, the used swap space is not a good indicator when trying to understand how much the wall time is affected. A better metric are the pages that are swapped in and out per time interval. In the Appendix, a comparison of the page swap activity of two jobs, see Figures A7 and A8, can be found. This highlights the difference between light and heavy swapping as explained in Subsection 4.6.3.

The explanation why the job duration increases is swapping, see Subsection 4.6.3. The restricted memory forces the system to swap pages from the memory to the disk, which is much slower. As a consequence, the CPU has to wait longer for input data that has to be retrieved from the disk, which increases the wall time. One conclusion that can be drawn from this plot is that the available memory for this job should not be below 5 GB of RAM, otherwise a slowdown in the wall time has to be expected. This is not true for every raw data reconstruction job, as with newer software versions or input data the memory requirement changes.

The profile of a job that heavily swaps pages in and out of memory can be found in Figure 5.14.

As expected, the disk read and write activity is heavily inflated, whereas the CPU usage drops significantly. After around 6000 s the disk is reading and writing at maximum capability and the CPU activity fluctuates between $0 - 70\%$. Half of the CPU activity after 6000 s is used by the system, that is swapping heavily, trying to handle the low amount of available RAM. These results on memory restrictions are important for the (memory limited) overcommittment, see Chapter 8.

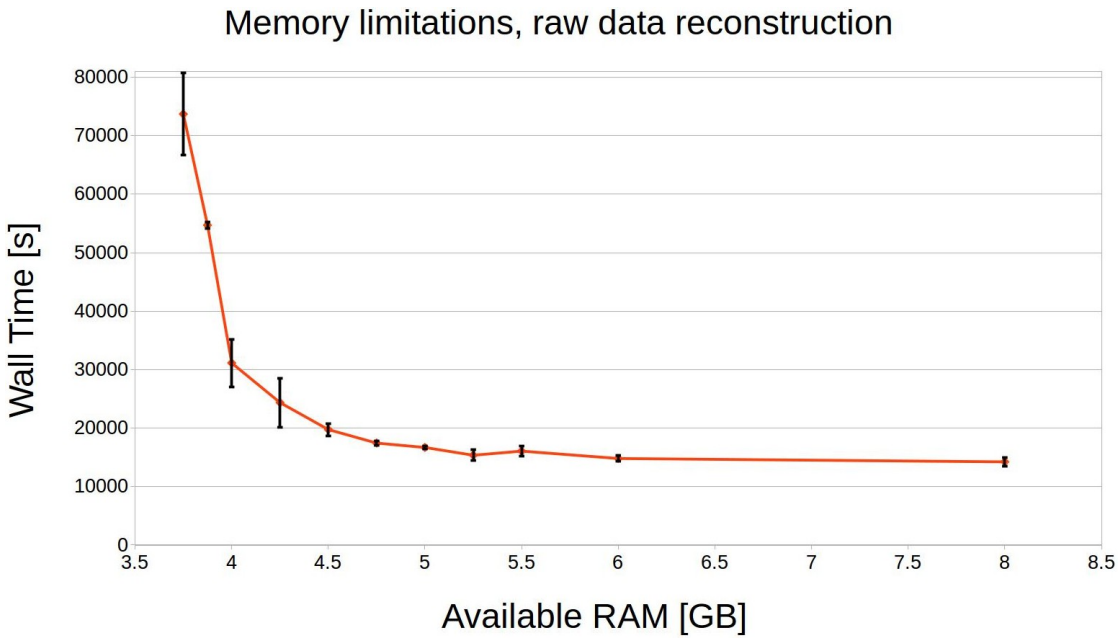

Figure 5.13: Average wall time of memory restricted jobs over four jobs using cgroups. This were four parallel processes on a four core VM. The input data was on the local disk. As can be seen from the error bars indicating the standard deviation (see Equation 5.1), the wall time fluctuates more, as soon as the jobs start to swap heavily. The data point at 3.875 GB RAM has a small error bar because it only consists of two instead of four measurements, which incidentally performed similarly. The measurement focused on the interesting region, when the memory limit impacts the wall time. Therefore there are no data points between $6 - 8$ GB of RAM. The job specifics are in the Appendix A.1.2.

Reconstruction fluctuation

The fluctuations of the wall time of the different workflows will become extremely important for the model later on in Chapter 6. The variation amongst repeating the *same* reconstruction job was of the same order, around 2.02% for CERN and 5.20% for Göttingen, as the *similar* simulation job variation. In Table 5.6, the *similar* job results are shown.

Figure 5.15 shows the wall time average as a function of an increasing number of *similar* reconstruction jobs.

The plot depicts the results for the VM in Göttingen, the results for the VM at CERN can be found in the Appendix, see Figure A6. Overall the fluctuations are small, in the order of a few percent, compared to the wall time.

The reconstruction workflow is the first workflow that uses a large set of input data.

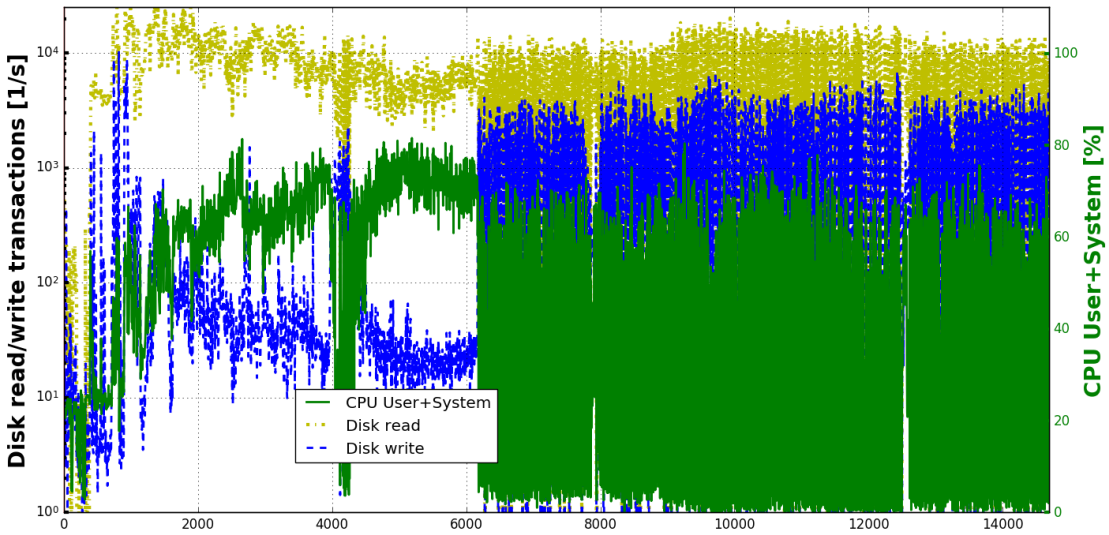

Figure 5.14: Profile of a memory restricted job. The input data were on the local disk. Note the logarithmic scale of the disk transactions.

	Average wall time [s]	Standard deviation %
CERN VM	14863	2.07
Göttingen VM	17043	4.92

Table 5.6: Summary of executing nine similar reconstruction jobs.

This is one explanation why the *same* job case already exhibits large fluctuations. These are introduced by having the disk as an additional component that can act as a bottleneck.

5.3.3 Simulated data reconstruction

The only difference between raw data reconstruction and simulated data reconstruction is that there are additional steps in the beginning, namely the pileup/digitisation, as can be seen in the diagram in Figure 5.16.

The higher the luminosity, the more interactions happen during a beam crossing. In 2012 there were, on average, 20 collisions per bunch crossing. Most of these are, however, inelastic proton-proton interactions, that are understood and therefore of no interest. These extra interactions are termed minbias events, whereas in the data these are called a pileup of 20. For a physics analysis, pileup can impact the choice of cuts or object selections. Also, efficiencies separating data from background may change, which is why the pileup of the data and the simulated data have to be the same. The luminosity will increase over the years and with it the pileup. This has an impact on the

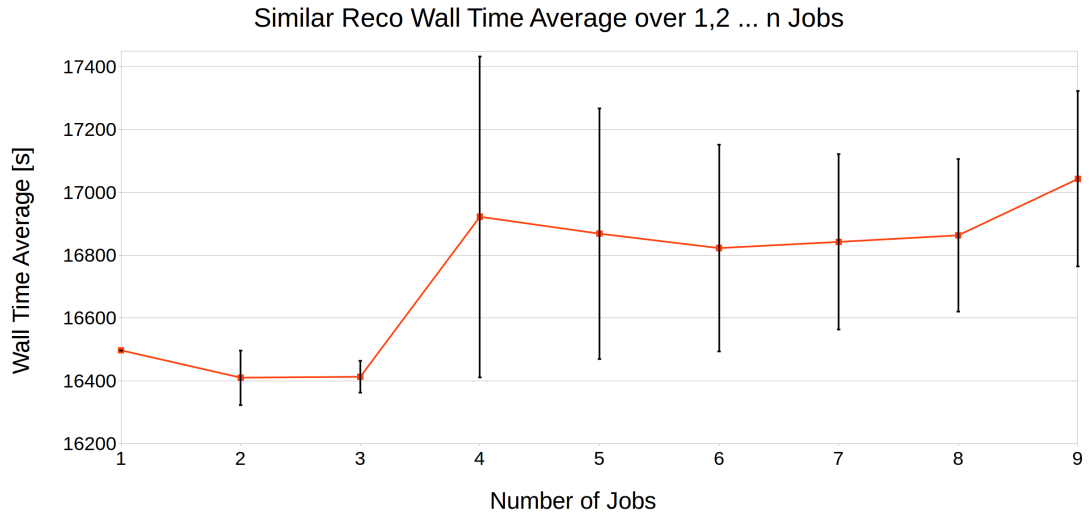

Figure 5.15: The wall time average over an incrementing number of *similar* reconstruction jobs, starting at one, at the VM at Göttingen. The black error bars show the standard error of the mean (see Equation 5.2). Note: in order to improve readability, the y-axis does not start at zero.

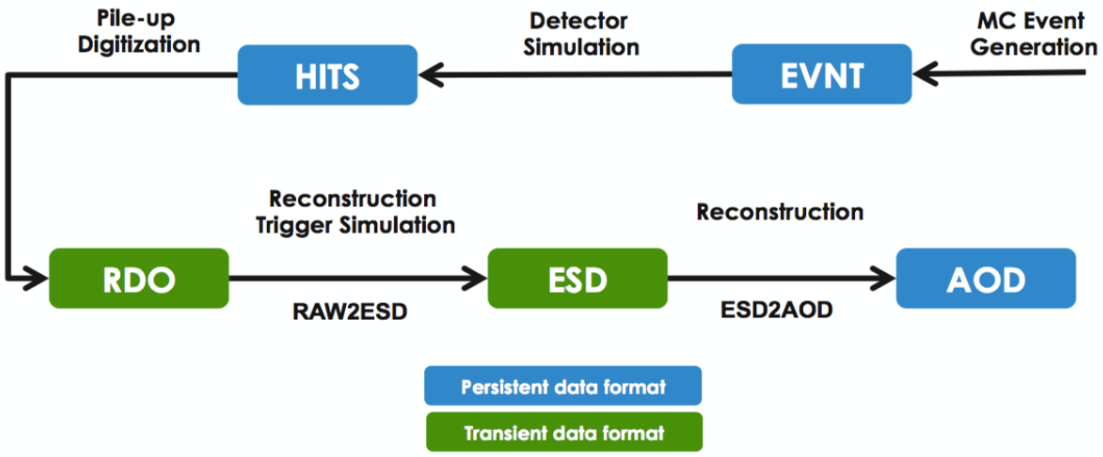

Figure 5.16: Simulation workflow chain with regard to the different data types.

MC simulation, because these additional minbias events have to be simulated. Since it would take too many resources to simulate the pileup every time, in the current model simulation leaves the minbias simulation out. The pileup is generated separately and overlaid afterwards.

Simulated data reconstruction profile

The profile is shown in Figure 5.17. The part after approximately 21000 seconds looks identical to the raw data reconstruction. The two additional steps RAWtoRDO and RDOtoRDOTrigger take up over two thirds of the whole job, from 0 s to 14000 s and 14000 s to 21000 s. In addition, the first transformation up until around 14000 s is reading much more data from disk, than is read afterwards, or in the other types of workflows.

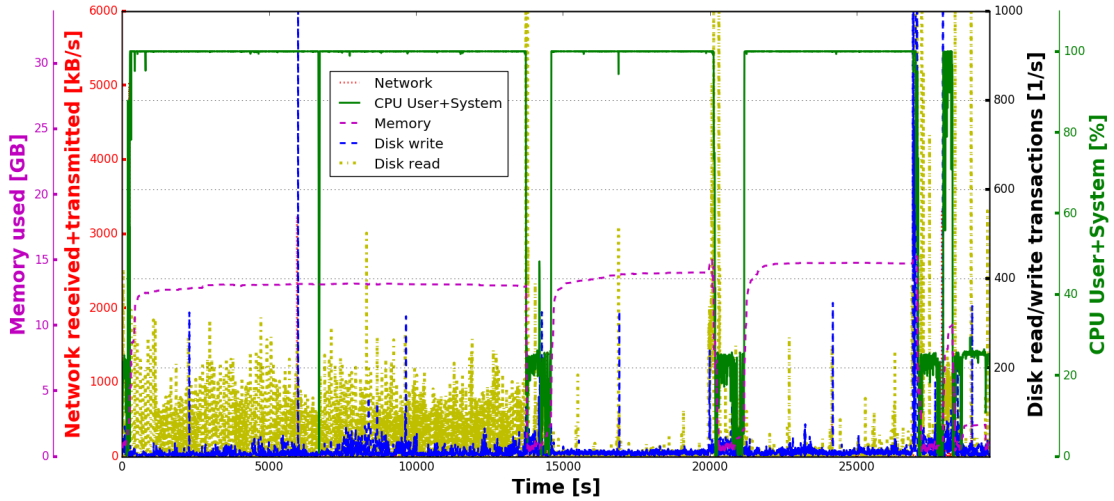

Figure 5.17: The digitisation and reconstruction workflow profile.

5.3.4 Digitisation

Digitisation transforms the simulated data, consisting of detector hits, into the detector response "digits" [110]. The detector response can range from a time range where a threshold in a certain sub-detector part was exceeded, to the signal profile. It mirrors how it would look for the real detector. The threshold is usually a current/voltage in a readout channel. This also takes into account the specifics of each subdetector component, like channel dependent variations or electronic noise [110]. The overall detector behaviour is taken from previous laboratory tests of the components, cosmic runs and test beam data. In addition the conditions for each run, such as dead channels and noisy rates, are read from a database and included.

As previously explained, there is not only one clean event in the detector at any time, all of the additional noise has to be included in the pileup [110]. Per bunch crossing, multiple proton-proton collisions happen. In addition, due to long signal integration times, additional background from adjacent bunch crossings can be registered. In the extreme case, for the muon chambers, at a bunch spacing of 25 ns, a window of around 70 bunch crossings has to be taken into account for the "out-of-time" pileup. This

background can be simulated and overlaid onto the simulated data from additional hits input files [110]. It scales linearly with the luminosity and the bunch spacing. There are also further inelastic proton-proton reactions and long lived particles within the detector, as well as beam-gas and beam-halo interactions that have to be taken into account [110]. The beam-halo originates from the beam interacting with accelerator elements, whereas beam-gas interactions are reactions of the beam with residual gas in the beam pipe. These interactions are taken from "zero bias" trigger data. It is reused with independent simulated data sets.

Digitisation and reconstruction fluctuation

In Table 5.7, the *similar* digitisation and reconstruction (DigiReco) job results are shown.

	Average wall time [s]	Standard deviation %
CERN VM	26142	0.83
Göttingen VM	2188	2.88

Table 5.7: Summary of executing ten similar DigiReco jobs. At CERN 2000 events were processed, whereas at Göttingen only 200 events were processed.

At CERN, ten times as many events were processed, which explains the large discrepancy between the wall times. Figure 5.18 shows the wall time average as a function of an increasing number of *similar* DigiReco jobs.

The plot depicts the results for the VM in Göttingen. Overall the fluctuations are small, in the order of a few percent, compared to the wall time.

5.3.5 Trigger simulation

Up to this part, the trigger, see Subsection 3.2.5, has not been considered for the simulated data. For a later analysis, it is important to stick as closely to the real data as possible, which is why the trigger has to be taken into account [113]. This will provide an accurate simulation of the selection efficiencies as well as the signal sensitivities. The trigger simulation takes the RDO data format as input. Since the trigger rates and cut-offs change in accordance with the luminosity and pileup, the software releases need to correspond to the releases used for the real data, unlike the MC simulation releases which evolve over time [113].

5.3.6 Reprocessing

Data reprocessing is basically a repetition of the reconstruction, which happens at a later point in time. Data is reprocessed due to a major improvement in the conditions status of the detector and/or software. Reprocessing applies, for example, updated calibrations, conditions, and improved tracking and corrects previously problematic regions of the detector. The data is reprocessed from the raw data format. The

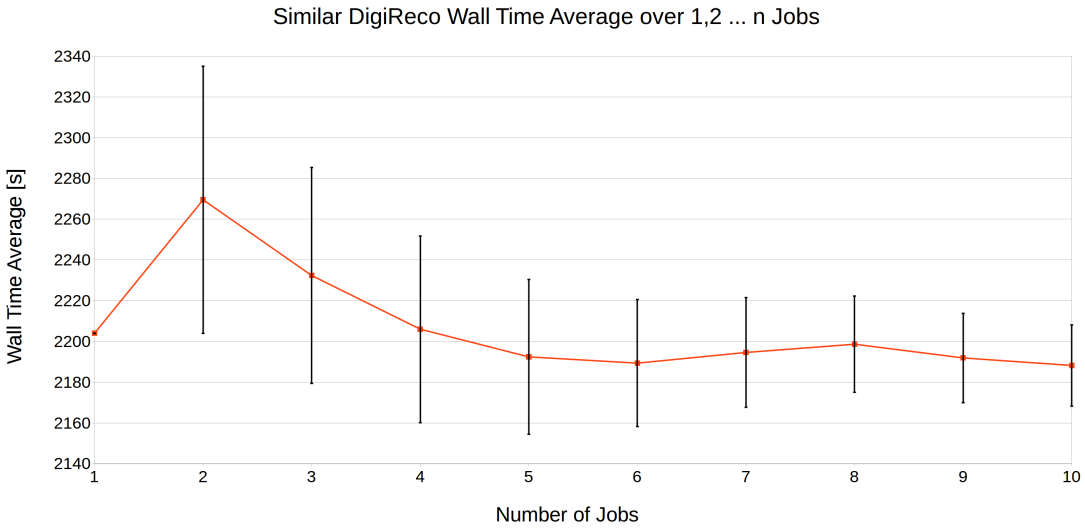

Figure 5.18: The wall time average over an incrementing number of *similar* DigiReco jobs, starting at one, at the VM at Göttingen. The black error bars show the standard error of the mean (see Equation 5.2). Note: in order to improve readability, the y-axis does not start at zero.

MC simulation data might also be reprocessed to give consistent reconstruction results. This is generally well defined and the resource requirements are known beforehand.

5.4 Analysis

In principle, a user can submit any piece of code, as long as the correct permissions are present. This means, even something that has nothing to do with a physics analysis[6] would be accounted as an analysis job. The fraction of non-physics jobs is probably very low, but it highlights the fact that the job behaviour can be hard or impossible to predict. Therefore and because the overall load of analysis jobs can vary widely, they are referred to as "chaotic" [114] as one analysis differs very much from another.

In general, however, an analysis job runs over large datasets and is therefore mainly I/O intensive. Mostly due to the small cross-section of most of the processes of interest, high statistics are required. High statistics means that the input data set has to be large. The decays/processes of interest can be very rare, meaning more data increases the chance of observing them in the first place. A nice example is the Higgs discovery, which did not happen moments after the LHC recorded the first Higgs-event, but after collecting and analysing several years worth of data.

[6] For example printing "Hello World!", or mining crypto currencies.

5.4.1 Group production

In the context of a group production, physicists within a group are interested in similar physics processes. Group production is a pre-processing step to the end user analysis, selecting specific tags and reducing the overall amount of data that has to be processed by individual analyses. It creates derived physics datasets, which are then used by the group members. This analysis step is more structured and easier to predict than an individual user analysis.

It mainly consists of the three steps: skimming, slimming and thinning.

Skimming means to reduce a set of ATLAS events to include only events of interest for subsequent analyses [87]. Depending on the use case, this reduction can be quite significant, reaching several orders of magnitude.

Thinning removes objects from within an event, based on the properties of the object [115].

Slimming is the process of reducing the event size by removing event information, such as variables, that are unnecessary for the analysis [116]. This is done uniformly and does not vary between different events.

5.4.2 Complete processing

Combining Figure 5.2 with the more in-depth information from the previous subsections gives a more complete picture, see Figure 5.19.

In principle, each of these steps can be run separately or everything can be combined into one job. This is handled differently between the different experiments. In ATLAS, pileup, digitisation and reconstruction are combined as previously explained.

Another important thing to note is that this graph does not only depict one sequence of workflows. There are many different analyses that require a plethora of different inputs. In the end, all of them require a plethora of different simulation, reconstruction and analysis workflows and combinations thereof.

Coincidentally, the classification of the different workflows agrees with the order of the workflows. Everything up to reconstruction, meaning the simulation chain excluding reconstruction in Figure 5.19, can be classified as CPU intensive as could be seen from their profiles. From the reconstruction onward, the workflows are I/O intensive.

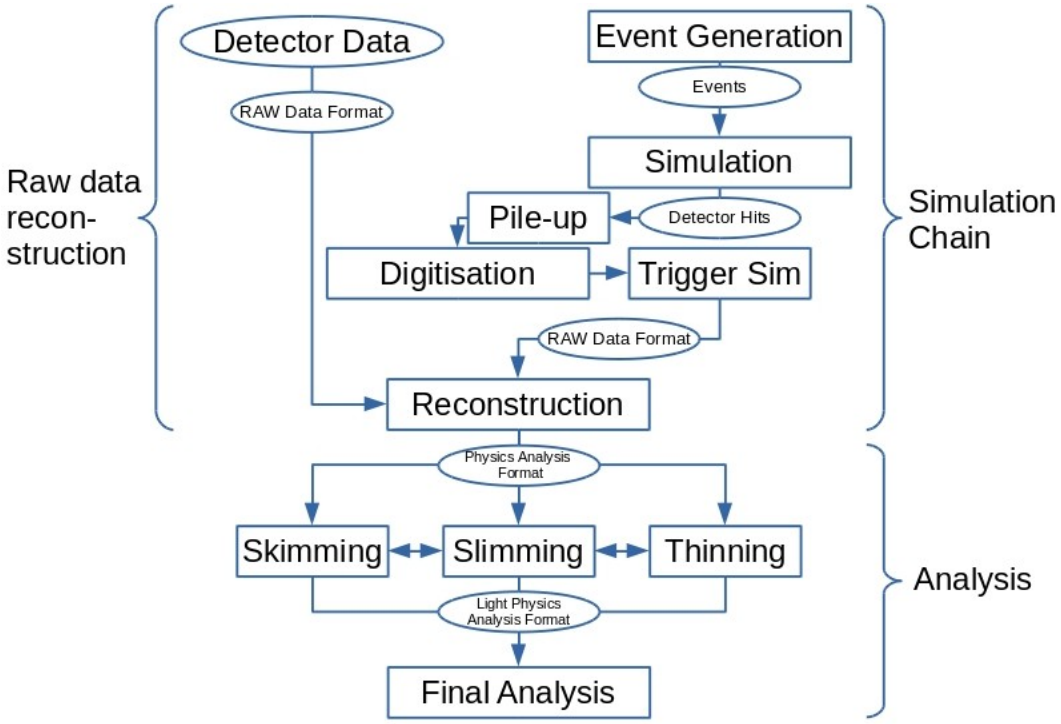

Figure 5.19: The whole data processing in detail.

CHAPTER 6

Models and predictions

"Prediction is very difficult, especially about the future." - Niels Bohr

In order to describe the workflow behaviour on an arbitrary infrastructure, a model was created. The goal was to make predictions that indicate the best workflow infrastructure combinations. The best infrastructure includes different Cloud offers.

The initial approach of the model can be seen in the paper published in the CHEP proceedings, see [117]. At that point, the idea was to split a workflow into very fine substructures which are the basis for the model. This, however, led to a plethora of input parameters. In order to reduce the number of input parameters, a whole new approach was used. This approach started with as few inputs as possible, to keep the complexity low. From there, input parameters were added successively once it was clear that the accuracy of the prediction could be improved significantly with this additional value.

There is a trade-off that has to be made, between keeping it simple and making it accurate. With this approach, the ideal model configuration has been found, making the model as simple as possible with a good predictive power. The benefits of a simpler model are that it is applicable without expert knowledge of the workflow and infrastructure and that it is more accessible to other experiments or users. Especially for Cloud computing, where some infrastructure aspects are unknown, a less complex model may be the only option. Disadvantages are that it might not be applicable to some special cases or configurations.

In the previous Chapter 5 an in-depth look at the profile of different workflows was given. These profiles are examples of how the workflows should look if they are run under good and stable hardware conditions. In reality, everything will be more chaotic due to constant changes to the software, input data and outside influences such as

neighbouring VMs with which resources are shared. The model is introduced to predict how different workflows will perform under these conditions.

The model is specifically tailored to make predictions within the Cloud, presented in Chapter 4.2, but the results and conclusions can just as easily be applied to the Grid. The significance of having a prediction already becomes apparent during the procurement. Here, the customer has to communicate their requirements to the Cloud provider(s). The correct balance between computing power, storage, input/output speed and memory has to be known. If the resources do not match the workflow requirements, money is wasted. It may be lost because too much of one resource was procured and the resources stayed unused, such as requesting too much network bandwidth for the needs. It can also be lost because too little of one resource was procured, in which case it becomes the bottleneck. This is then at the expense of the other resources that lie partly idle, e.g. too little network bandwidth and jobs (CPUs) waiting for input. The same is true for a Grid site.

To have a fair comparison, it is not only the workflow performance that is important, but also the optimal configuration of the site. The model is able to provide both.

In addition, an important question to be answered is, at which point using the Cloud is economically viable. A wrong understanding of the workflow behaviour in the Cloud can tilt this answer in either direction as the Cloud could be more/less performant than expected. This can potentially have a big impact in management decisions, e.g. whether to build a computing centre.

Furthermore, there is more than one type of workflow. Understanding how all of them would perform individually can help to decide which workflows to run on which infrastructure, even independently of the Cloud. On the Cloud, due to the hardware flexibility, this becomes even more attractive. Theoretically each workflow could run on a dedicated, specifically configured hardware.

6.1 Related work

There are different descriptive languages, such as TOSCA [118] and DADL [119], that describe application requirement and infrastructure resources. These were not adapted, as they were not flexible enough and because in the Cloud environment benchmarking is necessary. The descriptive languages match workflows to an infrastructure that fit their requirements. The model on the other hand, predicts the workflow performance on infrastructures that do not exactly match the workflow requirements, as is described in Chapter 8. This is necessary in order to find the optimal Cloud configuration, meaning to exploit the flexibility of the resource configuration. Additionally, optimisation techniques of the workflows could easily be investigated by the model, which would not have been possible using these descriptive languages. Furthermore, since some of the infrastructure parameters within a Cloud are unknown, benchmarking becomes necessary. The workflows can therefore already be described by their respective benchmarks and

thereby used by the model directly, making an additional descriptive language unnecessary.

Several models that predict workflows on different infrastructures already exist. Some of these concern themselves with different resulting metrics, such as the response time in [120], so they are not applicable. The more relevant ones, such as [121] and [122] concern themselves mostly with the memory access to predict the performance. When considering different memory limitations and swapping, these models become unreliable or very cumbersome.

In [123], a neural network is used to predict the application performance in a virtualised environment. This approach concerns itself with static infrastructures.

Further approaches are to model the entire infrastructure, such as in [124] and [125]. Modelling the whole WLCG is out of the scope of predicting and comparing different Cloud providers. In the end, focusing on the Grid as a whole and investigating how adding another Cloud site would impact the overall Grid is unnecessary. A Cloud site, can be different, with regards to the resources it provides and workflows it accepts, when compared to any of the existing Grid sites. A Cloud site that receives dedicated workflows, influences the overall Grid in a very minimal way.

Many of the models do not solve the problem of how to obtain the performance metrics for a Cloud site, or a generic benchmark is used. For Cloud sites not all the hardware and configuration information is available, in contrast to a Grid site. In the model introduced in this thesis, an approach that classifies and benchmarks the ATLAS workflows is used.

Another big difference to the presented tools is that they do not consider scenarios, in which the available resources are not enough. This means for example, they cannot explain what happens if a workflow is run on a machine that does not provide enough RAM, or how the performance is impacted in case CPUs are overcommitted, see Chapter 8.

The biggest advantage of the model in this thesis over the above presented models is, that they compute only the performance of one specific configuration. The possibility to find the optimal configuration of a Cloud site is an integral part of the model presented in this chapter. This allows the flexibility of the Cloud infrastructure to be considered, by matching the different workflows with the best corresponding infrastructure configuration.

6.2 The Workflow and Infrastructure Model

In order to predict workflow performances on different infrastructures, the "Workflow and Infrastructure Model" (WIM) was created. The model has been designed in a very general way, so that all ATLAS workflows, all the other experiments' workflows and even non-physics workflows can be described by it.

For the Cloud use case, there are several scenarios how the model can be used. The difficulty lies in the fact that when buying a CPU core, the performance for a given task is almost impossible to predict accurately, even if the manufacturer, generation and

clock speed are given. In addition, in a virtualised environment the provider can decide how many virtual cores run on any given machine. The provider could simply double the number of cores that are accessible to the customer, even though the underlying hardware remains the same, which would lead to a drop in performance. Therefore there is no way around benchmarking, meaning to measure the performance, see Subsection 4.6.1.

The first possibility is that the user has access to part of the Cloud infrastructures beforehand. In this case it is possible to run benchmarks and thereby acquire information on the performance of the Cloud infrastructures.

The second scenario is that there is no customer Cloud access. In that case, the customer has to supply a benchmark suite to the providers. The results are then published. This would usually already take place during the tendering phase of the procurement.

In either case, it is unavoidable to have infrastructure information from benchmarks, that can be used as input for the WIM.

The standard benchmark in HEP is HEPSPEC06, which could be used for this purpose, see Subsection 4.6.1. However, there are several downsides to the HEPSPEC06 benchmark, one of it being that it takes a long time to finish, in the order of one day, and is therefore very costly. Furthermore, there are very different workflows within HEP, some are better and some are badly represented by the HEPSPEC06 benchmark. For these reasons it was decided not to use the HEPSPEC06 benchmark as a normalisation in the WIM, but to use the actual ATLAS workflows that represent the workload instead.

6.2.1 Functionalities

A chronological view of the model shows that it was originally created to make a prediction on the required bandwidth of a future Cloud site. It evolved from there to include further prediction values. In order to compare Cloud sites, the overall output metric events/time/cost (ETC), which is measured in events/second/CHF, was chosen as a good comparative value. The abbreviation CHF stands for Swiss francs, the currency used at CERN. The ETC metric describes the "amount of physics", that can be computed per time and money. It solves the problem of comparing Clouds and different infrastructure configurations, as well as other optimisations, with each other.

Results

The WIM takes a set of input parameters and returns a predictive value. For different use cases, the overall model needs to be more flexible than just returning one possible output.

Any input metric can become the result, which is useful for cases where infrastructure requirements are unknown, e.g. the bandwidth, see Subsection 7.3.2. The model provides these results to choose from: ETC, events/cost, events/time, workflow duration, required bandwidth, cost/event, number of events produced in case of a fixed duration Cloud activity, download speed, total input size, download duration or overall CPU time.

Variations

The model has a built-in mechanism that provides the possibility to vary input parameters over a range. This then produces multiple results, one for each variation of the input.

Even multiple input parameters can be varied in parallel, leading to additional results. The best resulting value is provided, for example the one maximising the ETC ratio.

Correlations

Comparing these different results from varied input parameters with each other can easily show trends and the impact of the inputs.

Looking at trends can help to decide how to allocate resources, for example if the result increases or decreases significantly by increasing one infrastructure parameter. In a procurement, beneficial high-impact infrastructure aspects would then be favoured over low-impact ones, when tuning the hardware configuration.

A comparison can also highlight correlations between the different infrastructure and/or workflow aspects. This could, for example, indicate a possible bottleneck in one infrastructure aspect in the case where another aspect is varied.

Comparisons

Another application of the WIM, is the comparison of different Cloud sites. In this case the model is run once for each site or Cloud provider. This shows which site performs better in the given use case.

When searching for optimisations on an existing infrastructure, for example, cost does not play a role. Then the WIM can provide the events/s metric as output, which is a measure for the physics throughput and favours fast, usually more expensive hardware. If there is no time pressure, the infrastructure should be optimised for events/CHF, producing physics results as cheaply as possible. This favours cheap, usually slower hardware.

Plots

The model can vary input values over a range and display the results in a plot. In order to represent the changing results when input A and input B vary, a 3D-plot is generated. An example can be found in Chapter 8, see Figure 8.10. The input A would be on the x-axis, input B on the y-axis and the result on the z-axis. The maximum result value is given in the output and highlighted in the plot. This 3D-plot is colour coded for the z-value and can be rotated for better visibility.

In case more than two input values need to be compared, the model creates multiple of these 3D-plots, each of them with a different input C value. The corresponding input C values are indicated in the file name of the plots.

Since it would be tedious to look through all the plots just to get the best result, the overall maximum result value is given in the standard output, together with the

corresponding input values.

Overcommitting

Overcommitting is one of the possible optimisations that can be applied. It is described in detail in Section 8.1. In order to fully integrate overcommitting into the model, further input parameters have been added. With this functionality, that is described in Subsection 8.4.1, the model is able to indicate whether it is beneficial to run more parallel processes than there are available cores on a VM, at the cost of a higher RAM requirement.

Overview

With these powerful illustrative capabilities, the model is able to not only deliver the optimal hardware configuration, but also to show which input parameters are of lesser or higher importance for the overall outcome. This immediately indicates bottlenecks and the potential for optimisations.

The presented features provide answers to questions such as: there is a fixed budget, which Cloud infrastructure should be acquired in order to maximise the processed events? Which combination of bandwidth, caches and Cloud storage is the most beneficial? Similarly: what combination of workflows (combination of simulation, reconstruction and analysis) is the best to run on this Cloud?

The general use case includes:

- For an infrastructure and a workflow, how much bandwidth is needed?
- For a flat budget, how many events can be produced?
- Does this optimisation have a positive impact?
- How many events per second are produced?
- Does the event size affect the processing duration?

6.2.2 Model input

One of the WIM's goals is to describe how a workflow will behave within the Cloud. Therefore, the required input cannot be too specific, as some Cloud specific information is not publicly available.

A balance was struck, between accuracy and complexity, by creating a very basic model and carefully adding only relevant components to it. The relevance was determined by comparing results from the error and validation considerations.

The input parameters that were chosen in order to describe the workflow performance depend mostly on the benchmarks. In Figure 6.1, the different input and the most commonly used output parameters are depicted.

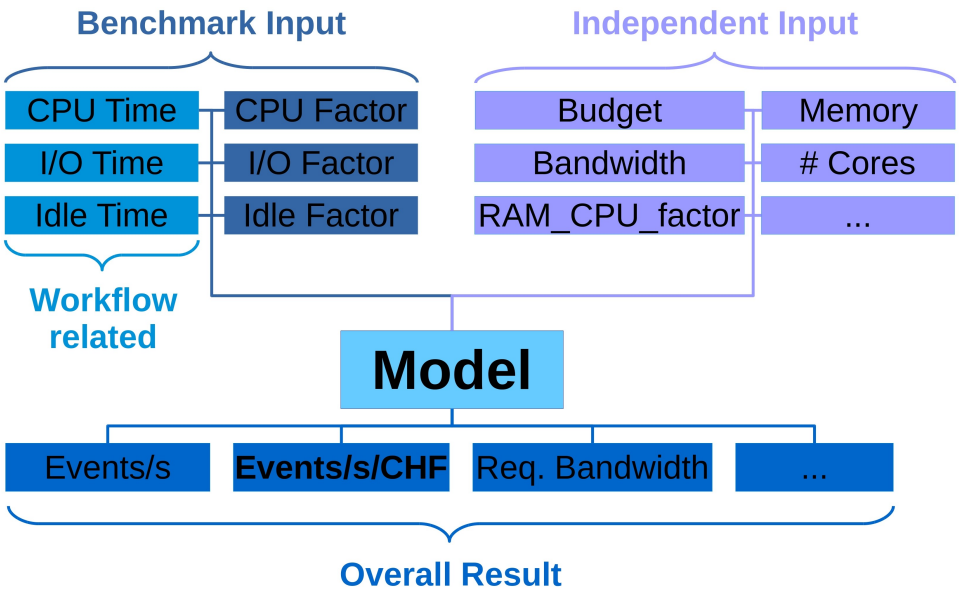

Figure 6.1: Model outline, depicting the different inputs and possible outputs.

On the bottom, the different possible outputs are shown. The list is not complete, as there are many more possibilities and combinations that can be obtained as a result. The left side depicts infrastructure and workflow related values that are obtained from the benchmarking. Due to the nature of the benchmarking these values are intertwined and cannot be separated into workflow or infrastructure inputs.

The values on the right side are either known or can be precisely specified during the procurement, so no benchmarking is necessary. The exhaustive list of input parameters that the model needs for the infrastructures, the workflow and the plot, can be found in the Appendix, see Subsection A.3. The central part labelled "Model", which contains the transformations of the inputs into the different outputs, is described in detail in the next Section 6.3.

The previously described functionalities in Subsection 6.2.1, describe how it is possible to compare the results of varied input parameters. This is achieved in the WIM by using lists as the variable input. An example is the default value of the number of CPU cores "Nr_Cores": [8.0, 9.0]. Having a list of input values as input for a variable can be done for at most three input values. The model will then create plots of all possible combinations.

Usage

The WIM is used in a simple way. First of all, in order to predict a workflow, the model needs to have some workflow parameters as input. These are obtained by running the

workflow on a reference machine and on the target hardware (e.g. Cloud). Running the workflow on the reference machine, where all the monitoring possibilities are available, provides the workflow specific metrics, such as the run time and the time the CPU was utilised. The hardware parameters, such as the CPU factor indicating the CPU power, are obtained by additionally looking at the results from the target hardware. Variables related to the reference machine are subscripted by $_{rf}$, variables related to the target machine are subscripted by $_{tg}$. Further details can be found in the appendix, see Subsection A.3.1.

6.3 Model logic

After seeing what the WIM can do, it is important to have a look at how it is done - to see the inner workings of the model. The most crucial part of the model is to determine the workflow duration on a given infrastructure. Most of the other outputs can then be derived from this duration.

A typical ATLAS workflow consists of several transformations as well as substeps, as could be seen, for example, in Section 5.3.1. Since the model uses the actual workflow as a benchmark, for basic usage it is not necessary to have detailed information about the intricacies of the workflow. These are looked at during the testing and validation in Chapter 7.

6.3.1 Workflow duration

The workflow duration prognosis is the central part of the model. It is chosen as a starting point, from which the underlying structure of the model can be explained in detail. The workflow duration is a complex combination of all the input parameters.

It can be divided into times when the CPU is used and times when it is idle. In the following subsections, first the CPU consumption time (Subsection 6.3.2) is explained and then consecutively all scenarios in which the CPU is idle (Subsection 6.3.4 to 6.3.6). Afterwards, an additional consideration is undertaken, that enters the equation when looking at multi-core systems (Subsection 6.3.7). Then the overall cost considerations are integrated in the number of resulting machines in Subsection 6.3.8. In the end, it is shown how the workflow duration is used to determine further final model results, see Subsection 6.3.9. The addition of the overcommittment functionality is done in Subsection 8.4.1.

At the topmost level, the workflow duration (WF_Time) is a linear combination of three different aspects, the CPU consumption time (CPU_Time), the CPU idle time ($Idle\_Time$), and the CPU I/O wait time (IO_Wait_Time).

$$WF\_Time = CPU\_Time + Idle\_Time + IO\_Wait\_Time \tag{6.1}$$

In order to understand some of the factors in the following formulas, it may be useful to understand in which format the input values are put into the model. As an example,

many input values are on a per event basis, which is why some results are obtained by multiplying by the number of events (Nr_Evts). More details can be found in the appendix, see Subsection A.3.

6.3.2 CPU consumption time

The CPU consumption time (see Equation 6.3) consists of the duration the CPU is in usage by a program. This is also called the processing time and processing happens all throughout the workflow.

The more powerful a CPU is, the more clock cycles it can perform in a given time and the more instructions it can perform per clock cycle. As a result, a program that is mostly CPU intensive will execute faster on a machine with a more powerful CPU. The CPU consumption time is therefore infrastructure and workflow dependent. How the CPU consumption time is obtained and what is exactly put into the WIM can be found in the Appendix, see Section A.3.

CPU power

In order to use the CPU consumption time on multiple infrastructures, it has to be normalised. This is done through benchmarks, that provide a measure for the relative CPU speed of two or more machines.

The reference machine, which can be any (virtual) machine that can run the benchmark, gets an arbitrary CPU power default value (CPU_power_{rf}). The model input for the CPU power of the target machine CPU_power_{tg} is obtained by:

$$CPU\_power_{tg} = \frac{CPU\_Time_{rf} * CPU\_power_{rf}}{CPU\_Time_{tg}} \tag{6.2}$$

This is the relative power of the target machine compared to the (arbitrary value of the) reference machine. Note that it is possible to use any kind of benchmark for this purpose, even HEPSPEC06 or non-physics workflows. According to this definition, a more powerful CPU gets a larger CPU_power value.

In the model, the CPU consumption time is easily calculated by multiplying the input CPU time with the Cloud infrastructure's CPU power, number of events and parallel AthenaMP processes.

$$CPU\_consumption\_time = CPU\_Time_{rf} * CPU\_power_{tg} * nr\_processes * Nr\_Events \tag{6.3}$$

Here, the multiplication with the number of parallel processes is important, because the model is created in a way, that more processes mean additional workload. Each additional process increases the number of processed events, which will be relevant in Section 8.1.

6.3.3 Idle time

The idle time, as the name suggests, is the part of a workflow, in which the CPUs are idle. This excludes time spans during which the CPU is idle because it is waiting on an I/O operation going through the block layer of the system. In general, CPUs waiting for network I/O operations are accounted in the idle time.

Other big contributors to the idle time can be found in multi-core VMs. At any point in time, if only one CPU core is used, the remaining cores count as idle. This is usually the case during the start-up phase, until all code is loaded into memory, and at the end of the job, when the different threads or processes finish at different times. In ATLAS jobs there are single-core merging steps that additionally contribute to the idle time.

The model treats idle time in a similar manner as the CPU time. Since benchmarks have already been run in order to obtain the CPU power, additional metrics from these results are used for the idle factor. The idle factor for the target machine ($Idle\_factor_{tg}$) is generated by comparing the idle time of the reference ($Idle\_Time_{rf}$) and target machine ($Idle\_Time_{tg}$):

$$Idle\_factor_{tg} = \frac{Idle\_Time_{rf}}{Idle\_Time_{tg}} \quad (6.4)$$

For simplicity, the arbitrary idle factor of the reference machine is ($Idle\_factor_{rf} = 1$).

6.3.4 I/O wait time

The I/O wait time refers to the time the CPUs are waiting for I/O operations to be performed. This includes all I/O that goes through the block layer, mainly to the disk or a network attached storage. Therefore the I/O wait time is mostly dependent on the disk speed. There are different kinds of disks, such as SSDs and HDDs. Some disks can handle more read/write operations per second. These are better suited to handle I/O operations, therefore positively influencing the I/O wait time. Of course there are also speed differences between the same kind of disks, albeit much smaller ones. If however, while one process is waiting on an I/O operation, another process could use the CPU in the meantime, it will. This will then not be accounted as I/O wait time. That is to say, there are scenarios in which the I/O wait time is close to zero but processes had to wait for I/O.

The model treats I/O wait time in a similar manner as the CPU and idle time, reusing the monitoring results from the previous benchmarks. The I/O power for the target machine (IO_power_{tg}) is generated by comparing the I/O wait time of the reference (IO_Time_{rf}) and target machine (IO_Time_{tg}):

$$IO\_power_{tg} = \frac{IO\_Time_{rf}}{IO\_Time_{tg}} \quad (6.5)$$

The I/O power of the reference machine is set to one ($IO\_power_{rf} = 1$).

6.3.5 Overhead time

There are some overheads that happen during a workflow. Their total duration is relatively short, therefore any improvements, such as better hardware, have only a slight impact on the overall workflow duration.

The most impactful overheads are the stage-in and stage-out duration. In the WIM it is intended to treat them separately from the idle time. The stage-in duration can be described by:

$$StageIn\_Time = \frac{Nr\_Events\_Files * Size\_Evt\_input}{Download\_Speed} \quad (6.6)$$

Where Nr_Events_Files is the number of events within the input files, $Size\_Evt\_input$ is the size of an event on disk. Since the model is not only describing one workflow, but an entire site, the overall download speed ($Download\_Speed$ [B/s]) is limited.

The total download speed of a worker node ($Download\_Speed$) is limited by the bandwidth ($Bandwidth$). The worst case can be determined from the available bandwidth and the number of parallel downloads:

$$Download\_Speed \leq \frac{Bandwidth}{Nr\_Machines} \quad (6.7)$$

$Nr\_Machines$ is the number of available (virtual) machines.

Since the time it takes to download the input is smaller than the processing time, there are two scenarios. The first scenario is that all workflows are trying to download at the same time. This can happen when the Cloud infrastructure is first initiated. Here the many parallel downloads are limited by the bandwidth and compete with each other. After some time, however, with some workloads finishing faster than others, the second scenario takes place. In it, the downloads of the workflows are distributed in time and only a smaller fraction of the downloads are overlapping.

A similar calculation is done for the stage-out time, replacing “input” with “output” and “download” with “upload”.

Another possibility is that instead of downloading the data, it could be read event-by-event during job execution. This is additionally negatively affected by high latencies, also known as the “event access time”. In this case, it is not feasible to separate the stage-in and stage-out from the idle time. The idle time would be used as it is, while bypassing the bandwidth considerations.

The remaining overheads can be summarised into a single input value. For most scenarios they can be set to zero, as all overheads should be included already. There is one exception, however, namely if a Cloud user creates and destroys a VM for every workflow. Since this happens completely outside and independent of the workflow, it has to be considered in addition.

6.3.6 Swap time

Swapping (see Subsection 4.6.3) is a special case of I/O wait, that is accounted as I/O wait time. An investigation of swap time is therefore only necessary when trying to describe the job behaviour in scenarios where too little RAM is available. This is for the most part relevant when doing overcommittment, see Chapter 8, and when changing the available amount of RAM within, for example, the Cloud.

There have been efforts to describe the swap time as a function of the swap space, but this is difficult and different for each transformation. An investigation into how the available RAM affects the wall time of a reconstruction workflow can be found in Subsection 5.3.2.

Since heavy swapping slows down the workflow significantly, this will never be a realistic option for an acceptable event throughput. This was illustrated in Subsection 4.6.5, where having an idle CPU core was preferred over having heavy swapping. Heavy swapping occurs if the available RAM (RAM) is smaller than the required amount (RAM_limit). Therefore, instead of describing the exact progression (non-linear), a large but constant penalty function is applied to the area where heavy swapping occurs:

$$Swap\_Time = \begin{cases} 0, & \text{for } RAM\_limit \leq RAM \\ 100000000000000, & \text{else} \end{cases} \quad (6.8)$$

This effectively cuts off any heavy swapping scenario from the viable results.

6.3.7 Undercommitted idle time

Undercommittment is another scenario that is implemented by the model, but can be ignored for regular usage.

The idle time when undercommitting (UC_Idle_Time) is only relevant for cases, in which less processes than cores are executed in parallel, for example, see Subsection 4.6.5. This effectively wastes CPU time in the amount of:

$$UC\_Idle\_Time = (Nr\_cores - nr\_processes) * Time\_One\_Workflow \quad (6.9)$$

which is accounted in the workflow time at the end. In general, the UC_Idle_Time is already accounted in the idle time. When comparing VMs with a similar core count and number of processes, it is also not needed.

It can be useful when the reference machine has more cores than the target machine, which is not recommended. Then, in order to get comparable results, one would run as many parallel processes on the reference machine as there are cores on the target machine. The UC idle time will then help to compare the results.

The overall duration of the workflow $Overall\_WF\_Time$ is calculated by applying undercommittment from Equation 6.9 to the standard workflow time:

$$Overall\_WF\_Time = (WF\_Time - UC\_Idle\_Time)/Nr\_Cores \quad (6.10)$$

For regular workflows, the UC idle time is zero.

6.3.8 Number of machines

According to the different use cases, there are several possibilities of how to determine the resulting amount of machines. The important final metric is how many machines are available over which period of time ($Nr\_machines$).

In the Cloud scenario, the assumption is, that there is a budget ($Total\_Cost$) that cannot be exceeded and a given time frame in which the Cloud activity takes place ($infrastructure\_duration$)[1]:

$$Nr\_machines = Total\_Cost/machine\_cost * infrastructure\_duration \quad (6.11)$$

When increasing/lowering the amount of RAM per core (from RAM to RAM_new), the number of machines decreases/increases (from $Nr\_machines$ to $Nr\_machines\_new$). The new number of machines is calculated with the help of an exchange rate between CPUs and RAM (RAM_CPU_factor). This exchange rate is Cloud provider specific and depends on their pricing scheme.

$$Nr\_machines\_new = (Nr\_Cores - (RAM\_new - RAM) * RAM\_CPU\_factor) * Nr\_machines/Nr\_Cores \quad (6.12)$$

Alternatively, the infrastructure may already be owned by the user, in which case the number of machines does not have to be computed and can simply be put into the model. In that case, the infrastructure duration becomes obsolete. The overall duration of the exercise in the Grid case is determined by the workload.

6.3.9 Final result

There are different metrics that are meaningful. The model can output several metrics, as they are just different combinations of the already calculated values. The most useful metric, the events/time/cost value ETC is obtained by the following equation:

$$ETC = (Nr\_Events * OCF) * Nr\_machines\_new/Overall\_WF\_Time/Total\_Cost \quad (6.13)$$

where OCF is the overcommit factor that will be introduced in Subsection 8.1.1. For the general case without overcommitting $OCF = 1$. The metric events/time can be obtained by removing the $Total\_Cost$ from Equation 6.13.

The schematic of how the inputs are computed up to the final result can be found in Figure 6.2.

[1] The infrastructure duration cannot be smaller than the workflow duration.

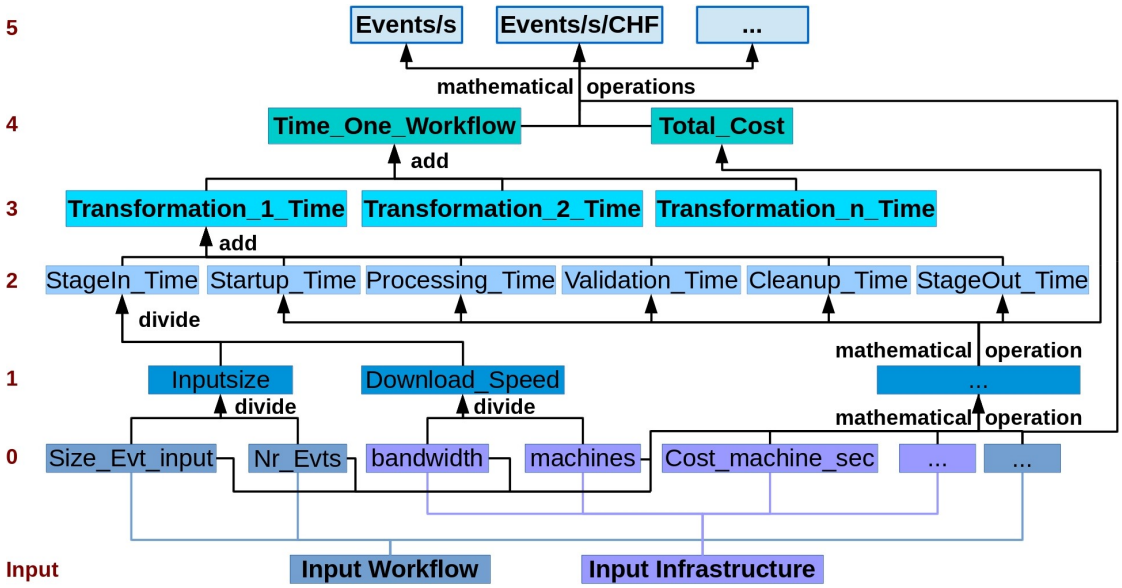

Figure 6.2: The model calculation schematics. The red numbers on the left side indicate the level [117].

The calculation starts at the bottom with the input values, some of which are displayed on level zero. These are combined to higher level input metrics, which are shown in level one. In level two, combined workflow and infrastructure metrics are created. These are combined to represent the duration of the different workflow substeps in level three. A combination of those, gives the wall clock time of the entire workflow. In parallel to this, the cost considerations in level four were computed, using inputs from level zero and one. The final result in level five is obtained by combining the workflow duration with the cost.

6.3.10 Estimation of uncertainties

The WIM provides an uncertainty to each of its results, that indicates the accuracy of the prediction.

The variations of each workflow can be measured in detail, which was done in Section 7.1.3. In the case of Cloud computing, the infrastructure underlies further fluctuations. This is dealt with by considering the standard deviation on the target infrastructure, thereby including workflow as well as infrastructure fluctuations. The CPU time, I/O wait time and idle time (x_i) each have a standard deviation (s_i). These get propagated up to the final result, which automatically takes care of the impact they have on the final result R:

$$R = R(x_1, x_2, ...) \tag{6.14}$$

$$s_R = \sqrt{\sum_{i=1}^{m} \left(\frac{\partial R}{\partial x_i} s_i\right)^2 + 2 \sum_{i=1}^{m-1} \sum_{k=i+1}^{m} \left(\frac{\partial R}{\partial x_i}\right) \left(\frac{\partial R}{\partial x_k}\right) s(x_i, x_k)} \tag{6.15}$$

In this case $m = 3$, corresponding to the count of x_i. This formula includes covariance terms ($s(x_i, x_k)$) that become zero if the two values are uncorrelated. This is the case, except for the CPU_Time and the $Idle\_Time$ for which the exact correlation is unknown. An upper bound of the uncertainty is estimated through:

$$s_R = \sqrt{\left(\sum_{i=1}^{m} \frac{\partial R}{\partial x_i} s_i\right)^2} \tag{6.16}$$

6.4 Programming tools

The model was realised in Python v2.7.12. The modules that were used, were numpy v1.11.0, json v2.0.9, math, pandas v0.17.1, unittest and matplotlib v1.5.1.

6.5 Complex workflow model

Initially, in order to give a good prediction, a job was split into several transformations. These were put into the model separately.

Transformations were treated independent of each other and put together afterwards in a modular way, because different jobs and workflows can consist of different combinations of these transformations. An example would be DigiReco and raw data reconstruction, which both use the same reconstruction transformation, but also transformations that cannot be found in the other workflow. Ultimately, a job or a workflow, consisting of multiple jobs, are both treated as a series of their consisting transformations.

As mentioned at the beginning of this chapter, this approach has been abandoned. A positive side effect is that the possibility to use several transformations as input still remains. This now provides the functionality that a mix of several workflows can be entered at the same time into the model.

CHAPTER 7

Cloud workflow modelling

Over the course of this thesis, several commercial Cloud procurements took place within CERN IT. The most recent pre-commercial procurement tender, called HNSciCloud, took place within the HelixNebula initiative. Most notably, this EU project provided access to several commercial Cloud providers and consortia. With access to multiple providers, a good understanding of the Cloud environment and its behaviour could be experienced and measured, see Section 7.2.

In order to make statements about the Cloud, especially comparisons between the providers, the WIM is used. Before using the WIM, that was described in detail in Chapter 6, it is imperative to understand the Model's accuracy. This includes the inherent uncertainty estimation that is provided by the Model. This chapter therefore first explores the parameter space of the WIM, while comparing its predictions to independent measurements, see Section 7.1

Two of the Cloud providers offered an object storage. The possibilities to make use of these, was however very limited. The results are explained in Subsection 7.2.2.

7.1 Validation

7.1.1 Strategy

In order to do a precise validation, the strategy was to start the validation from a simple controlled environment and to increase the complexity step-by-step. The validation is only meaningful, if it is ensured, that different combinations of workflows and infrastructures will be predicted correctly. The schema can be seen in detail in Figure

7.1.

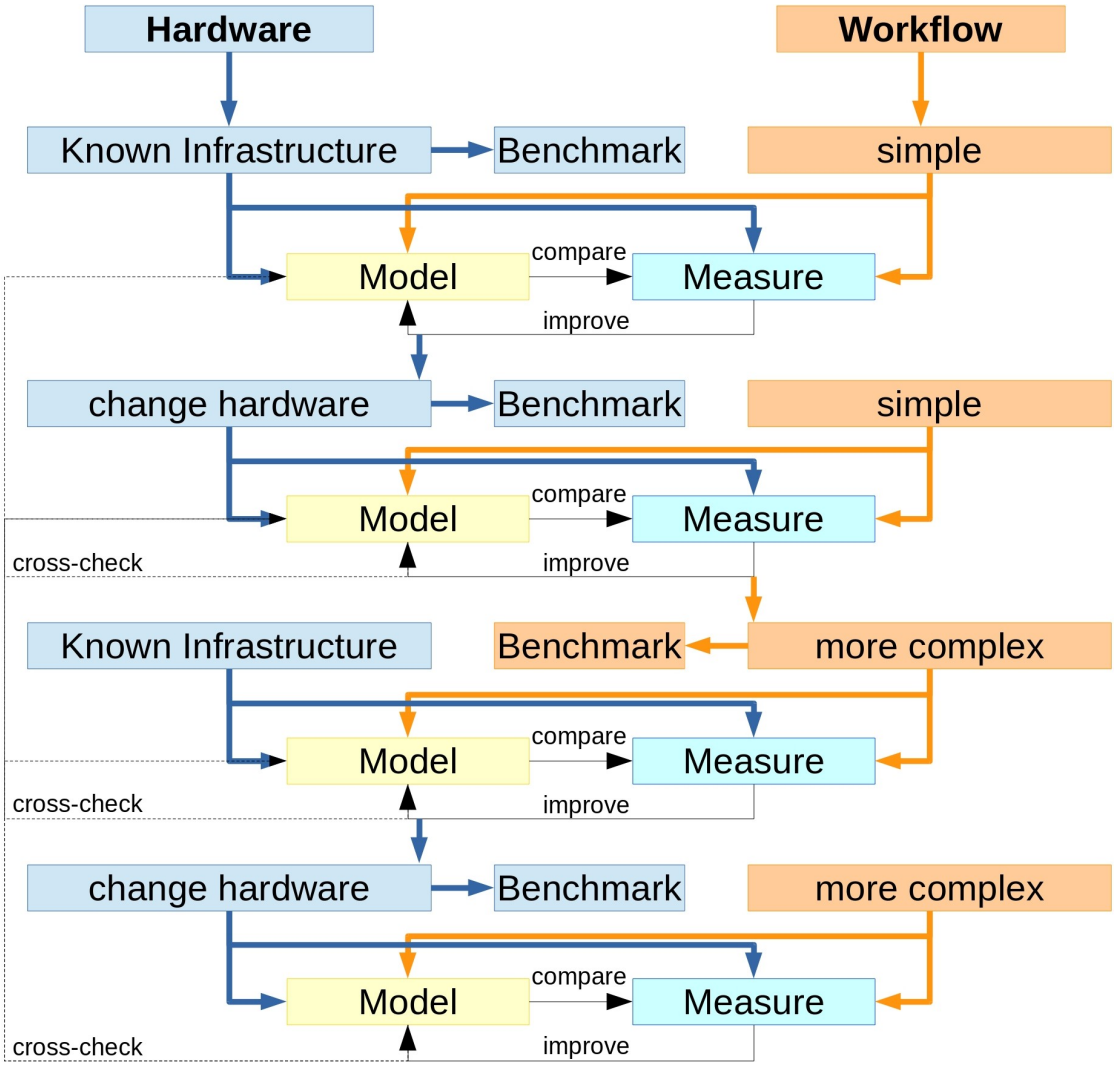

Figure 7.1: Detailed plan of how the validation was performed. The starting point is at the top and the validation progresses towards the bottom. In reality, this flow would continue further down. The diagram would have been longer, but for space purposes it has been cut off at the end, since the principle can already be understood from this diagram.

The validation begins on a well-defined, simple workflow on a well-known infrastructure. Even though the infrastructure is already known, it is benchmarked to provide a precise Model input.

Then, hardware and workflow aspects are changed alternatingly. This is indicated by a different coloured downward arrow, either on the left or right side. Each change in

the configuration requires a new benchmarking iteration. The different colours of the benchmarks in Figure 7.1 indicate, whether the same benchmark is used to describe hardware or workflow aspects.

After every step, the Model results are compared to the measurement and, if necessary, improved. Necessary in this case means, that the difference between the results is large. The improvements range from removing software bugs, to adding workflow or infrastructure parameters, to increase the accuracy. Each improvement requires the results of all previous configurations to be cross-checked. Initially, an overall accuracy of 20% was aimed for. After seeing the results, this goal became more ambitious.

After reaching the bottom of the diagram, the validation is not finished. Further workflow and hardware parameters can be varied, following the same principle. In reality the steps were parallelised as much as possible in order to save time. This meant having several different hardware configurations on different VMs at the same time and processing the tests in parallel.

The workflows that were used, were ATLAS workflows. An increase in complexity was achieved by going from the simplest to the most complex workflows. The workflows that were chosen in the above described order were: event generation, Monte-Carlo simulation, raw data reconstruction, and simulated data reconstruction. Event generation is the simplest workflow, because it does not use input data and creates very little output data. It is therefore CPU bound and also the only workflow that runs on only a single core. Monte-Carlo simulation uses only the output generated by event generation as input, which is very little. It is therefore also CPU bound, using multiple cores. Raw data reconstruction is more data intensive than Monte-Carlo simulation. Simulated data reconstruction, including digitisation and pileup, is the most data intensive of the investigated workflows.

Direct validation

Before this structured approach, a more direct validation had been attempted. In it, the WIM results were immediately compared to Cloud results within T-Systems. This T-Systems Cloud procurement happened before the HNSciCloud activity.

The direct validation approach skipped over the small iterations, in which the complexity should be increased step wise. In the diagram in Figure 7.1, it would mean to start from the bottom most step.

The T-Systems Cloud was integrated into the ATLAS production system as an additional Grid site. In order to account for the differences to a Grid site, not all job types were allowed to run on the T-Systems infrastructure. The workflows that were run, were taken from the ATLAS job queue. This means, no specific tests were performed on this infrastructure, which makes a comparison more challenging.

In order to decrease the parameter space, the job variety for the model was limited by looking only at ATLAS reconstruction jobs. In order to have a point of reference and a possibility to compare, a further condition was that these are similar jobs, running on both the T-Systems Cloud and at CERN.

While this gave an indication that the Model is able to describe the job behaviour on these infrastructures within a reasonable error of 20%, it was not clear where this error originates from and what influences it. The complexity of the non-homogeneous infrastructures and the reconstruction workflow did not allow for a thorough investigation. This becomes clear, when attempting to improve the Model. A very simple example would be, that the overall wall time of the workflows is underestimated by the Model. By modifying the time it takes to download the input data, this can be compensated, but what if in truth this discrepancy comes from a slower disk? In this case, a look at the actual download duration of the jobs can indicate that this compensation is incorrect, but other factors such as slower CPUs within the infrastructure can be harder to detect. In Subsection 7.3.3 the results for the above mentioned T-Systems activity can be found.

7.1.2 Setup

In order to give a prediction, the Model needs input information about the workflow and the target infrastructure. In the following, there is an overview of parameters that have been varied during the testing. Each of these variations can manifest itself in multiple Model input parameters.

The input parameters are obtained from benchmarking. In the following sections, actual ATLAS workflows are chosen as benchmarks. The range of different *jobtypes* for ATLAS is sufficiently narrow, that not too many benchmarks have to be performed. The results are more precise than when using a generic benchmark. The downside is that they are also more specific to the types of benchmarks used, meaning each *jobtype* needs its own benchmark.

Workflows

The workflow variations happened on three different levels, which were applied in the tests. First, all *different jobtypes*, namely event generation, Monte-Carlo simulation, reconstruction, digitisation and pileup jobs are covered. On the second level, software versions and the corresponding input data from different time periods are varied. The third and bottom most level covers *similar* jobs, meaning the variation of input data within the same ATLAS run or fill.

Infrastructure

The infrastructure differences are covered by running on many different VMs. The most precise measurements were achieved by using VMs that were located on dedicated

hardware located at CERN and Göttingen. These two VMs were hosted on machines that no other users had access to, so there was no influence from neighbouring VMs, known as noisy neighbours". True independence of these machines is guaranteed by the physical separation of these machines. The specifications for these VMs can be found in the Appendix, see A.7.2 and A.7.1.

Initially, there was a 4-core VM at CERN and an 8-core VM at Göttingen, but as later results show, it is sub-optimal to compare machines with different numbers of CPU cores. Therefore, all measurements were re-done with an 8-core VM at CERN as well. Some more specifications can be found in the appendix, for the 4- and 8-core CERN VMs see Subsection A.7.2, for the 8-core VM at Göttingen see A.7.1. More information on the specifications of the particular CPUs of the two VMs can be found in the Appendix 2 and 6. The reason why eight cores were chosen is because it is the most commonly used ATLAS multi-core setup, see Figure 7.2. The many single-core jobs are event generation jobs as well as individual merging jobs and specialised jobs such as trigger reconstruction. It also includes the many ganga test jobs, that check whether a site is functional. In addition almost all analysis jobs are single core jobs.

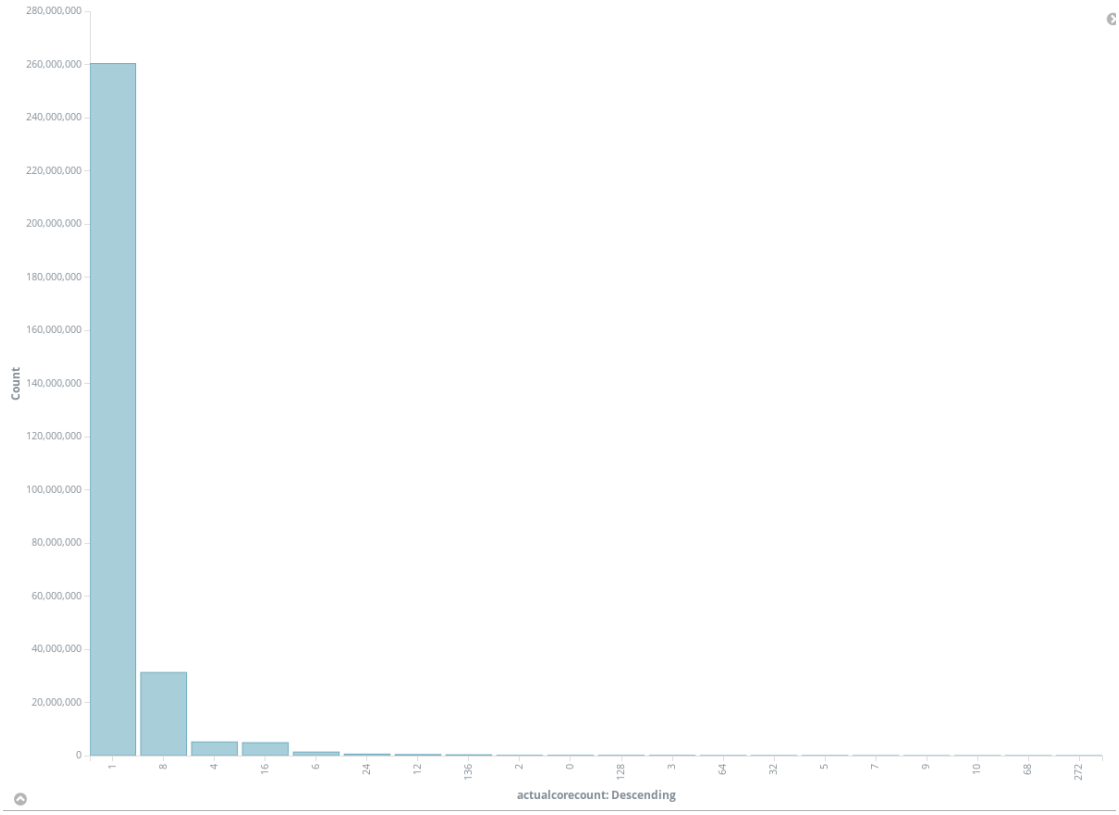

Figure 7.2: Cumulative number of jobs per core configuration over one year.

Further infrastructure variations result from using VMs from commercial Cloud providers. There is an overlap with the investigation into the Cloud, as the results can also be used to validate the Model.

Through HNSciCloud, access to three different Cloud providers was granted. Due to the nature of these resources, it was difficult to obtain the specifications of the underlying hardware. The three providers were: Exoscale, IBM and T-Systems. Their machine specifications can be found in the Appendix, see Subsections A.7.3, A.7.4 and A.7.5.

There was no control over what the provider and other customers did on the underlying hardware and neighbouring VMs.

Initially, the tests were performed on only one VM at T-Systems, one VM at IBM and three VMs at Exoscale. These tests took place from the end of September until the middle of October. The conclusions about Cloud computing that could be drawn from the single-VM scenario were limited. It was therefore mainly used in order to validate the Model, the results can be found in Subsection 7.1.4.

At a later stage, from the middle of October until the end of November, additional resources became available. This increased the amount of VMs to a total of 10 on each provider. A better understanding of the job behaviour on commercial Cloud providers could be gained from the larger scale tests. These can be found in Subsection 7.2.1.

Some providers were not very flexible with the amount of RAM per VM they could offer. If the desired configuration was unavailable, one with too much RAM was chosen. The standard 2 GB RAM per core configuration of the grid was then achieved by limiting the RAM on the machine artificially. This was realised by executing a program, that reserved a block of memory the size of the difference between the existing and the desired configuration. The source code of the program can be found in the Appendix, see 1.

Monitoring and accounting

The different metrics used in this investigation were obtained from several sources. First of all, the ATLAS workflows themselves have a built-in monitoring functionality, whose results can be obtained from the log files (mem.full.*, mem.summary.* and jobReport.json) and the standard output. In order to get a more granular monitoring, the 'sar' commands of the sysstat package were executed every 5 seconds, see Appendix 3. Additional metrics are collected via the /proc and /sys pseudo-filesystems that provide interfaces to the system kernel, see Appendix 4 and 5. Since the 'sar' and '/sys' metrics refer to the entire system, not just the processes of interest, it was ensured that no other processes influence these measurements. Nothing else was executed and all automatic 'cron' processes were disabled.

7.1.3 Validation - workflow fluctuation

When measuring or even when running the exact *same* workflow on the same machine at a different time, the duration of the processing will experience statistical fluctuations.

Moreover, a set of *similar* workflows will show additional fluctuations regarding its duration. Both of these types of fluctuations have been investigated in Chapter 5.

The workflow behaviour will also differ when moving to newer software and input data, as well as when looking at *different jobtypes*. It is important to understand these systematic differences as the model will have to consider these either by adapting the input or by including them in the error estimation. In the following, they are eliminated by re-benchmarking for all *different jobtypes*. Some jobtypes were split further into additional benchmarking categories.

This was done for cases in which the resource usage pattern within the jobtype showed large variations and a discrepancy between the infrastructure input parameters was seen. These large variations can be easily found by looking at the discrepancy between the WIM prediction and the measurement. If the discrepancy becomes larger than 10% it is useful to try and benchmark anew. If the new results are improved significantly and a discrepancy between the infrastructure input parameters can be seen, a new benchmarking category should be used.

Instead of this phenomenological approach, the ATLAS workflows have an inherent separation into different jobtypes. These jobtypes roughly classify the resource usage patterns. Some additional differences enter, when different transformations are used.

An investigation into covering the different workflow behaviour not by re-benchmarking, but by an error estimation has been undertaken in Subsection 7.3.1. These numbers highlight how the discrepancy quickly surpasses the 10% threshold, when combining very different jobs into one benchmark.

The WIM predicts the run time of a large ensemble of jobs, giving a prediction for the mean duration. In fact, it is assumed that the amount of jobs that will be run, for example on a Cloud, will average out most fluctuations that can be observed in individual workflows. This is confirmed for all workflows by the convergence of the averages in Figures 5.6, 5.8, and 5.15.

However, the WIM accuracy in the end still suffers from fluctuations within the workflows. This is due to the fact, that reference workflows are used in order to benchmark the infrastructure and to describe the behaviour of future workloads. Therefore, before any statements about the accuracy of the Model can be made, a closer look at the workflow behaviour and its fluctuations has to be undertaken. This was done for each individual workflow in the corresponding Subsections 5.2.1, 5.2.2, 5.3.2, and 5.3.4. For the WIM, only the fluctuations among *similar* jobs are relevant, as they represent what is done in a real production environment.

The effect of the fluctuations is lessened by benchmarking multiple times, but they cannot be completely discarded. If the input values deviate too much from reality, the predictions of any Model will differ from reality. How often the benchmark should be run, for sufficiently good input values for the Model, can be answered by Figures 5.6, 5.8 and 5.15. Judging from the plots and the low fluctuations therein, it is enough

to run a small number of event generation/simulation/reconstruction jobs in order to get good input values for the Model. In principle, the little amount of fluctuations would justify benchmarking with only one job. In order to be sure that the first job is not an outlier of some kind, throughout this thesis ($n = 3$) jobs have been used for the benchmarking. Using at least this many jobs is generally recommended. Another reason for multiple benchmarks are the uncertainties which are estimated from the fluctuations of the benchmark, see Subsection 6.3.10.

The goal of the WIM is to have an estimation that agrees with reality at a better than 20% level. A successful validation shows that the predictions do not deviate more than 20% from the measurements.

7.1.4 Results

In this subsection the validation results are presented. This validation of the WIM was done by predicting the wall time of a set of jobs on different VMs. The predictions are then compared to the results of subsequent measurements.

This was performed as follows. The reference VM at CERN was used to obtain the workflow input parameters. Three benchmark jobs per workflow were then performed on different target VMs located at Göttingen, IBM, T-Systems and Exoscale. On Exoscale three different VMs were use: EXO 1, EXO 2, and EXO 3.

Afterwards, the Model is used to predict how subsequent workflows will perform. Finally, further workflows are executed on these infrastructures and the wall time is compared to the Model prediction.

In order to correctly interpret the results, it has to be kept in mind that there are workflow fluctuations. These were described in Sections 5.2 and 5.3.

Event generation

The resulting Model prediction for the event generation workflow agrees well with reality, as can be seen in Table 7.1.

The biggest deviation in wall time was found on the EXO 3 VM. Since the event generation workflow is dominated by CPU processing, the large deviation originates from that component.

This also explains why a deviation in the I/O wait time of over 60% for the TSY VM has a small impact on the wall time, which is predicted with over 99% accuracy. The fraction of I/O Wait time compared to the wall time is very small.

Monte-Carlo simulation

There is a good agreement between prediction and measurement, as can be seen in Table 7.2. The CPU factor, CPU, Idle and IO wait times have been obtained from 25 reference

Model Discrepancies [%]	Wall Time	CPU Time	I/O Wait Time	Idle Time
IBM	**0.15**	0.95	0.78	0.95
TSY	**0.84**	1.48	-66.14	1.50
EXO 1	**-0.34**	0.72	16.92	0.85
EXO 2	**0.02**	0.86	-18.13	1.08
EXO 3	**3.00**	3.97	-8.35	4.13
GOE	**0.68**	0.90	-2.17	0.84

Table 7.1: Deviation of the Model prediction from the measurement of similar event generation jobs. The wall time has been split into its components in the last three columns to 5. The benchmark was performed three times, the results are the average over 23 jobs.

Monte-Carlo simulation jobs run at CERN and three reference jobs run at the respective provider.

Model Discrepancies [%]	Wall Time	CPU Time	I/O Wait Time	Idle Time
IBM	**2.49**	2.58	-1.56	3.01
TSY	**1.98**	0.96	7.28	6.83
EXO 1	**2.63**	1.92	2.02	5.77
EXO 2	**1.83**	1.12	11.29	4.53
EXO 3	**2.68**	1.47	16.95	9.13
GOE	**2.49**	1.86	3.03	1.11

Table 7.2: Deviation of the Model prediction from the measurement of similar Monte-Carlo simulation jobs. The benchmark was performed three times, the results are the average over 23 jobs.

Trying to model the simulation with event generation input factors resulted in much worse results, with a large deviation between the prediction and the measurement. This is described further in Subsection 7.3.4.

An attempt to find a constant conversion factor between the event generation and the simulation CPU factor has been undertaken unsuccessfully. The factor changed too much between different workloads.

Reconstruction

The reconstruction workflow includes more I/O operations than event generation and Monte-Carlo simulation workflows. The results can be found in Table 7.3.

Since reconstruction is more data intensive, the IO wait and idle durations become more relevant.

Model Discrepancies [%]	**Wall Time**	CPU Time	Idle Time	I/O Wait Time
IBM	**1.85**	0.22	4.12	15.16
TSY	**-0.40**	-0.11	0.22	-2.83
EXO 1	**-0.17**	0.18	-0.50	4.37
EXO 2	**0.51**	0.26	1.53	1.85
EXO 3	**-0.05**	-0.07	0.61	3.70
GOE	**1.02**	0.17	1.66	3.29

Table 7.3: Deviation of the Model prediction from the measurement of similar reconstruction jobs. The benchmark was performed three times, the results are the average over 10 jobs

The results of the Model agree on a 2% level, however there are some larger discrepancies in the wall time components. Especially the I/O wait time shows a large deviation for IBM in relative numbers.

In absolute numbers, however, the I/O wait time is comparatively small, see Table 7.4. This simply shows that small fluctuations have a high impact in the relative numbers. For the model validation these discrepancies can therefore be disregarded.

Durations	**Wall Time [s]**	CPU Time [s]	Idle Time [s]	I/O Wait Time [s]
IBM	**4849**	17225	17740	3281
TSY	**4330**	18531	15473	501
EXO 1	**3144**	13080	12012	56
EXO 2	**3107**	13253	11462	62
EXO 3	**3145**	13358	11662	62
GOE	**5437**	25532	17394	669

Table 7.4: Absolute numbers of the job duration measurement from similar reconstruction jobs. The results are the average over 10 jobs.

The high idle time can be explained by the single core merging steps. The reconstruction profile in Figure 5.9 shows, how during this merging only one of eight cores is active. The merging therefore quickly accumulates idle time.

Uncertainties

In Table 7.5 the deviation of the Model prediction to the measurement is compared to the prediction of the uncertainty. How the model uncertainty is propagated was described in Subsection 6.3.10. The input values are obtained from the three benchmarks, especially their fluctuations.

What can be seen, is that the prediction of the uncertainty fluctuates a little between the different providers, but differs significantly between different workflows. This means that the workflows themselves are the limiting factor for the prediction in this scenario.

[%]	Event Generation		Simulation		Reconstruction	
	Error	Uncertainty	Error	Uncertainty	Error	Uncertainty
IBM	0.15	1.56	2.49	4.23	1.85	4.34
TSY	0.84	0.74	1.98	4.46	-0.40	0.35
EXO 1	-0.34	1.12	2.63	4.36	-0.17	0.62
EXO 2	0.02	1.78	1.83	4.33	-0.51	0.93
EXO 3	3.00	1.03	2.68	4.43	-0.05	0.94
GOE	0.68	1.11	2.49	4.41	1.02	0.73

Table 7.5: The columns labelled "Error" show the already seen wall time deviation [%] between the measurement and the Model prediction. The columns labelled "Uncertainty" show the percental uncertainty that is estimated by the Model and provided together with the prediction.

It must be kept in mind, however, that these are single VMs.

Another observation that can be made, is that the range of the uncertainty does not always include the measured deviation, e.g. Event Generation / EXO 3. This is to be expected, as the uncertainty estimation is not an upper boundary. As most deviations lie within the given uncertainty, this highlights that the uncertainty estimation is a good and useful indicator.

7.1.5 Conclusion

In conclusion, for single VMs within different Cloud providers, the WIM is sufficiently accurate. This can be seen from the fact that the model deviations do not exceed 5%, which is well within the 20% goal. It should be kept in mind, however, that the WIM is dependent on the benchmark.

In addition to the accuracy, the estimations of the uncertainties agree well with the deviations of the Model predictions.

For these reasons the WIM is deemed complete and ready for usage. The WIM is applied in different scenarios in Section 7.3.

7.2 Cloud measurement

Before applying the model to the large scale Cloud procurement, the different providers are investigated in detail. The HNSciCloud procurement at CERN provided access to VMs at three different commercial Cloud providers, namely: Exoscale, IBM and T-Systems. In the following, different (previously introduced) workflows are run on the VMs of these providers. For this real life Cloud exercise, there is no knowledge of which VMs share the same hardware, how many there are, or what workload is run on them. Furthermore, the conditions are not stable, meaning it can be possible that any of the infrastructure parameters suddenly changes during a measurement. These

measurements will provide valuable information of how the workflows behave in this kind of environment.

In addition, insights into whether the WIM predictions are still valid, or whether the uncertainty estimation has to be adapted, are obtained.

7.2.1 HNSciCloud: large scale

The single-VM measurements were already presented along with the Model validation. In the later stages of the prototyping phase, additional resources became available. A total of ten VMs per provider, with eight CPU cores per VM, were provisioned. This increased the scale of the tests significantly to 240 CPU cores.

There was no information available on which VMs were collocated on the same underlying hardware. Therefore resource contention among these ten VMs was possible, as all tests were run in parallel on all VMs.

The same workflows have been run on all providers, see Tables 7.6, 7.8 and 7.7. The in-depth details of the underlying workflows, with which the tests could be repeated, can be found in the Appendix in Section A.6. On a coarse level, the workflows that have been run correspond to the different ATLAS job categories, namely: event generation, simulation, reconstruction and digitisation. A more fine grained division has been undertaken for the reconstruction workflows. This was done because reconstruction is data intensive and therefore more complex, so additional parameters could be varied. Each job category was repeated at least ten times per VM. This means, that for each job category around 300 measurements were done.

Due to various reasons, individual jobs failed. It was attempted to re-run each failed job. Time constraints limited the amount of repetitions of failed jobs. After this became clear, more than ten jobs per category and per VM were run. The total amount of measurements per provider and job category therefore stayed above 100, even with job failures. The exception is Reco 2, with slightly fewer jobs. Due to the failures, the averages that are presented below, can consist of a varying number of underlying measurements.

Exoscale

In Table 7.6, all Exoscale results are summarised, together with their standard deviations. The wall time is split into its components.

From the previous measurements it is already understood, that the different job categories represent a diverse mixture of jobs. This is reflected in the measurements.

A difference between the measurements over ten VMs on a Cloud provider compared to single VMs in a controlled environment, see Sections 5.2 and 5.3, can be observed. The standard deviation is increased for the larger setup. This is due to the fact that on top of the fluctuations that were already observed, there are infrastructure fluctuations. These result from differences among the VMs and the more volatile environment.

Exoscale	Wall Time [s]	CPU Time [s]	Idle Time [s]	I/O Wait Time [s]
EvGen	**2915 ± 137**	2875 ± 171	20179 ± 980	4 ± 1
MC Sim	**1279 ± 61**	8489 ± 434	1669 ± 244	10 ± 1
Reco 1	**5737 ± 440**	29524 ± 2225	15293 ± 2046	732 ± 134
Reco 2	**5700 ± 190**	26222 ± 748	18519 ± 1203	489 ± 290
Reco 3	**4547 ± 478**	18672 ± 2142	17235 ± 1877	184 ± 57
Reco 4	**4528 ± 398**	18578 ± 1874	17185 ± 1529	180 ± 25
Reco 5	**3147 ± 88**	13250 ± 410	11731 ± 594	54 ± 6
Reco 6	**3529 ± 451**	16909 ± 2267	11173 ± 1563	101 ± 91
Reco 7	**5550 ± 943**	29660 ± 5587	13801 ± 1610	725 ± 646
Digi Reco 1	**1210 ± 123**	4945 ± 214	4568 ± 928	30 ± 5
Digi Reco 2	**8381 ± 681**	55780 ± 4814	10495 ± 912	121 ± 15

Table 7.6: In this table all test results of the jobs run on the Exoscale infrastructure are shown. Each line constitutes similar jobs. Jobs that took less than 500 seconds CPU time are excluded, as these failed.

One example would be the different performance of the VMs as illustrated in Figure 7.3. What can be seen is that VMs 5 and 7 take much longer to complete the same workload than the other VMs. Furthermore, the other eight faster VMs differ slightly in their performance as well. These kinds of differences in the performance of VMs can have different root causes, which are not necessarily stable in time. In addition, different workflows may be more or less sensitive to these differences, meaning the impact varies.

Single jobs also appear within VMs that deviate largely from the others. The three cases can be found in VM 1 job number 10, in VM 5 job number 3 and in VM 8 job number 3. These outliers represent an additional type of short termed fluctuation.

In Figure 7.4 the same VMs as in Figure 7.3 are used.

Ignoring the fact, that a different workflow is run, what can be seen immediately is that now, only VM 5 is taking longer to finish the workload. VM 7 that was slower than VM 5 in Figure 7.3, is now performing at the level of the other 8 faster VMs. This drastic change in performance hints at a change to VM 7 that lay outside of the influence or knowledge of the Cloud procurer. Luckily for the procurer, the two figures are in chronological order, so this change represents an increase in overall computing power.

This shows, that including volatile hardware can have a negative impact on the stability of the wall time, making predictions more difficult and more error-prone. Assuming that the provider offers better hardware as a bonus, the model can however predict a lower bound for the event throughput.

IBM

The overview over the IBM results can be found in Table 7.7.

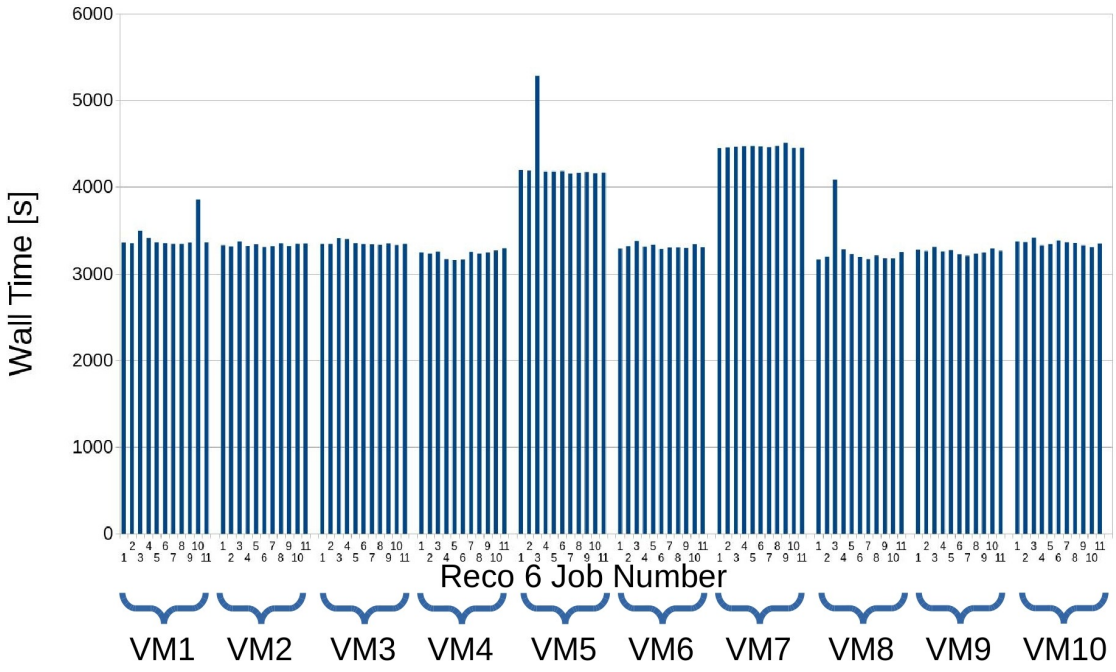

Figure 7.3: Wall times [s] of all Reco 6 jobs executed on Exoscale. The jobs are organised according to the VMs they ran on and in the same order.

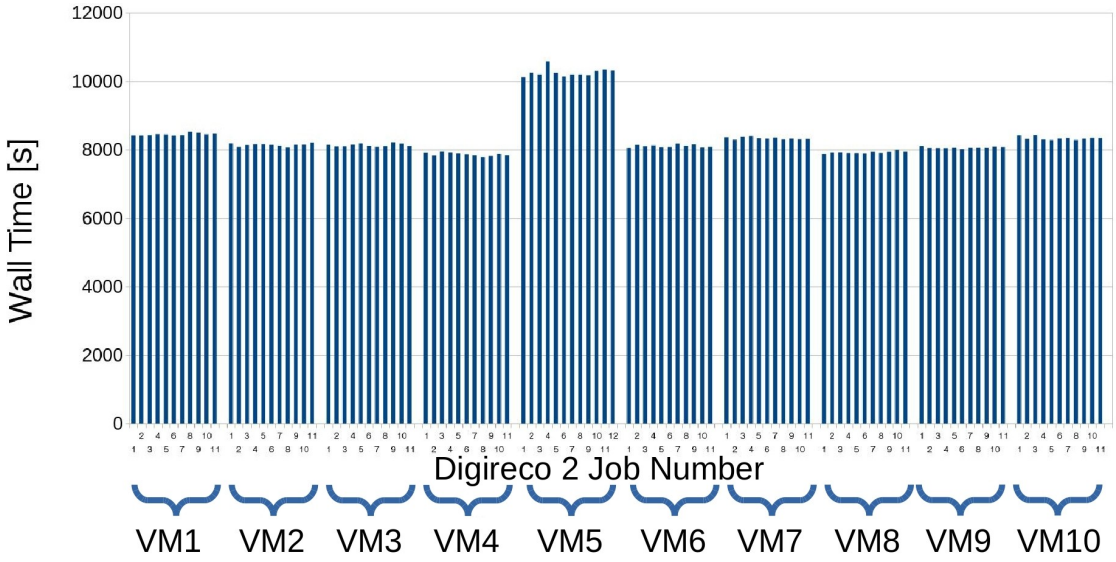

Figure 7.4: Wall times [s] of all Digi Reco 2 jobs executed on Exoscale. The jobs are organised according to the VMs they ran on and in the same order.

IBM	Wall Time [s]	CPU Time [s]	Idle Time [s]	I/O Wait Time [s]
EvGen	**3927 ± 118**	3906 ± 120	27263 ± 850	9 ± 1
MC Sim	**2321 ± 353**	16184 ± 2709	2352 ± 324	11 ± 2
Reco 1	**8193 ± 571**	43588 ± 4073	19600 ± 876	2065 ± 641
Reco 2	**8165 ± 598**	37346 ± 2776	26573 ± 2008	848 ± 130
Reco 3	**7061 ± 629**	30786 ± 5083	23708 ± 3257	617 ± 120
Reco 4	**7009 ± 604**	29713 ± 4002	25223 ± 1605	643 ± 101
Reco 5	**4756 ± 680**	20352 ± 3155	17214 ± 2501	257 ± 79
Reco 6	**4919 ± 468**	25036 ± 3562	13943 ± 645	247 ± 59
Reco 7	**7630 ± 760**	39786 ± 5063	19953 ± 1519	836 ± 148
Digi Reco 1	**2019 ± 172**	8690 ± 1145	7276 ± 358	61 ± 14
Digi Reco 2	**15116 ± 2395**	104959 ± 18841	15155 ± 846	439 ± 75

Table 7.7: In this table all test results of the jobs run on the IBM infrastructure are shown. Jobs that took less than 500 seconds CPU time are excluded, as these failed.

The VM performance is lower than in Exoscale. What can be seen is that different job types are impacted more or less by the differences between the two infrastructures. Some workflows are significantly slower, such as Digi Reco 2, whereas others are not slowed down as much, such as Reco 7.

The differences in fluctuations are shown and described in detail later.

The IBM infrastructure was found to be heterogeneous as was seen before for Exoscale in Figure 7.3.

T-Systems

One difference between the T-Systems VMs and the VMs from the other providers is that eight of the T-Systems VMs were located on the same underlying host. This was achieved by choosing a "dedicated host" as the underlying infrastructure for these VMs in the T-Systems interface.

The T-Systems results are summarised in Table 7.8.

In comparison to what has been seen for Exoscale in Table 7.6 and for IBM in Table 7.7, the wall times are increased. An interesting fact is that different job categories are affected differently by the difference in the infrastructure. The slow-down factor between the wall times from Exoscale and T-Systems, differs between the job categories. There it ranges from around 1.4 for event generation to around 3.75 for simulation.

The fluctuations are also higher, as is shown and discussed in detail later in Table 7.9.

T-Systems	Wall Time [s]	CPU Time [s]	Idle Time [s]	I/O Wait Time [s]
EvGen	**4089 ± 126**	4098 ± 126	28394 ± 885	15 ± 25
MC Sim	**4808 ± 1298**	22863 ± 3930	3944 ± 1227	78 ± 74
Reco 1	±	±	±	±
Reco 2	**15430 ± 2612**	59086 ± 9645	34790 ± 3420	14318 ± 3326
Reco 3	**8681 ± 1827**	32646 ± 5262	29511 ± 4847	5061 ± 4525
Reco 4	**10785 ± 1534**	42030 ± 4942	31941 ± 3296	3876 ± 3162
Reco 5	**7099 ± 1203**	28419 ± 3597	18479 ± 2247	387 ± 62
Reco 6	**7986 ± 1810**	34097 ± 4936	20870 ± 5020	1712 ± 3241
Reco 7	**15348 ± 2886**	61478 ± 11789	27058 ± 5703	17861 ± 3272
Digi Reco 1	**2789 ± 524**	10497 ± 1546	7588 ± 1200	208 ± 53
Digi Reco 2	**25821 ± 6151**	168476 ± 16656	23207 ± 5077	1441 ± 573

Table 7.8: In this table all test results of the jobs run on the T-Systems infrastructure are shown. Jobs that took less than 500 seconds CPU time are excluded, as these failed.

Figure 7.5 highlights the heterogeneity of the infrastructure that was also observed for Exoscale.

One issue that can be spotted when looking at Figure 7.5, is that there appears to be a pattern. This wall time pattern, can be seen throughout VMs 2-8 and VM 10. The series of jobs was started at the same time and the VMs in question were located on the same host.

The pattern manifests itself especially in the second job, which took longer than the neighbouring ones. In addition, from the third job onward, the wall time seems to be rising steadily. Due to the almost flat wall time distribution in VMs 1 and 9, the possibility that the pattern originates from the differences between the jobs can be excluded. This has also been double-checked by repeating the same jobs on different VMs. No pattern of this kind has been found in the other VM's wall time distribution.

Due to the role as customer, the possibilities to investigate the origin of this pattern were very limited. One possible factor that played a role in creating the pattern, is that the VMs may have contended for a hardware resource. This could have impacted the different workflows differently. Another explanation would be an outside interference, such as a neighbouring VMs. Finally, it could also have been a combination of the two, or something entirely different.

Comparison

First of all, for all three providers the wall times fluctuated more using multiple VMs compared with single VMs. The standard deviations of the wall time within the three providers are compared in Table 7.9.

The different workflows themselves seem to behave more or less stable in their duration.

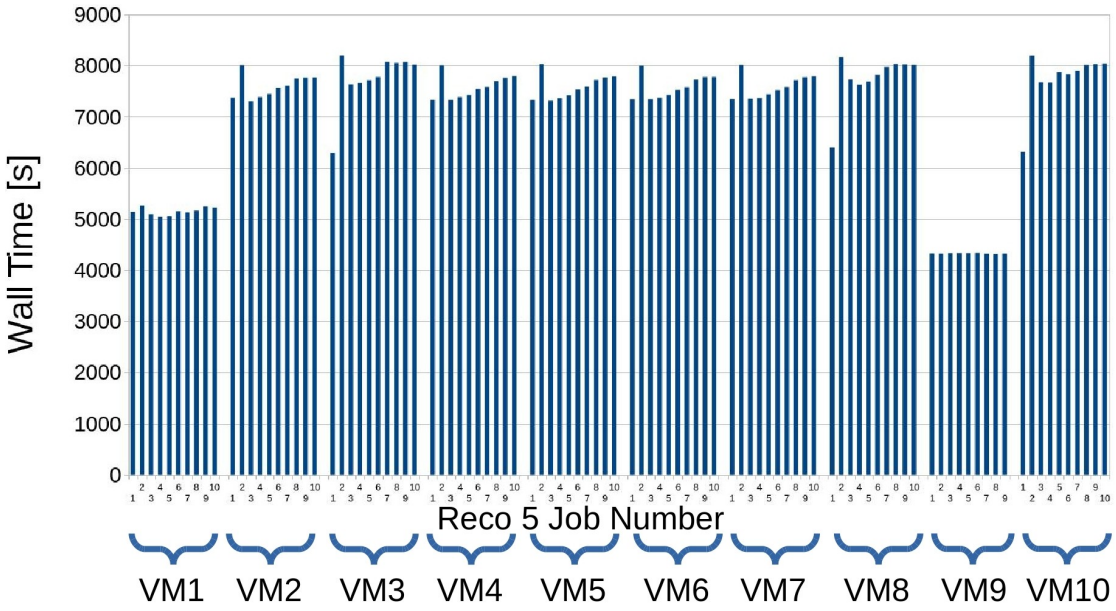

Figure 7.5: Wall times [s] of all Reco 5 jobs executed on T-Systems. The jobs are organised according to the VMs they ran on and in the same order.

For example, EvGen has a lower standard deviation than Reco 6 throughout all three providers. It has to be kept in mind however, that the tests took place at different periods in time. A change in outside influences, such as the VM speed-up that was demonstrated in Figures 7.3 and 7.4, could have taken place between and during the tests.

EvGen seems to be impacted the least by the VM heterogeneity. This was not investigated in detail, but the EvGen workflow differs the most from the other workflows. It can be seen from the fact that it is the only workflow that does not have the built-in functionality of running on multiple CPU cores. Modern CPUs are very complex, making it probable that the different workflows do not touch on the same CPU aspects and optimisations. Especially for code that is less modern and less optimised for specific CPU functionalities, the differences in a heterogeneous mixture of CPUs can appear smaller.

The numbers in Table 7.9 also indicate differences between the providers. Exoscale and IBM appear to fluctuate less than T-Systems. An important thing to keep in mind is that a high standard deviation is not necessarily a reflection of an unstable infrastructure. This can be made clear from Figures 7.3, 7.4 and 7.5. There appears to be a mixture of more and less powerful VMs, which result in a large standard deviation.

Due to the independence of the workflows running on the different VMs, a large variation in performance is not negative. What counts in the end is to have a high

Wall Time	Exoscale StdDev [%]	IBM StdDev [%]	T-Systems StdDev [%]
EvGen	4.70	3.02	3.07
MC Sim	4.77	15.19	27.00
Reco 1	7.68	6.97	
Reco 2	3.34	7.32	16.93
Reco 3	10.50	8.90	21.05
Reco 4	8.78	8.62	14.22
Reco 5	2.79	14.31	16.95
Reco 6	12.77	9.52	22.66
Reco 7	16.99	9.95	18.80
Digi Reco 1	10.21	8.52	18.77
Digi Reco 2	8.12	15.85	23.82

Table 7.9: Comparing the Clouds. Displayed are the standard deviations of the wall time. A comparison between the wall times can be found in the Appendix in Table A2.

event throughput. Therefore a diverse and difficult to predict infrastructure with high throughput is preferable over a homogeneous infrastructure with a low throughput.

A high variability can however lead to a low performance, if the provider did not account for it correctly. This could be the case if the provider guarantees a computing power that is equal to the average of the infrastructure. Depending on how the VMs are provisioned, the customer could get 'unlucky' and therefore get less than the agreed upon performance.

Different hardware types or generations within the infrastructure make an accurate prediction of the overall performance more difficult. This can be seen in Subsection 7.3.3, where the model results are shown.

Taking into account that there are two different generations of hardware reduces the standard deviation in the wall time, as can be seen in Table 7.10.

A comparison between these results and the ones in Table 7.9 show that the large spread that is reflected in the standard deviation of T-Systems decreases. The IBM results are impacted less, when splitting them into two categories. The biggest impact can be found in Digi Reco 1 and 2 (IBM and T-Systems) and in MC Sim (T-Systems). The reason why the T-Systems standard deviation seems to benefit more from this split, is because the difference between the faster and regular machines is bigger than for IBM.

7.2.2 HNSciCloud: object storage

In Table 7.11 the last three T-Systems Digi Reco results are summarised.

The last three digitisation measurements are T-Systems specific, because IBM did not

Wall Time	IBM fast StdDev [%]	IBM StdDev [%]	T-Systems fast StdDev [%]	T-Systems StdDev [%]
EvGen		3.02		3.07
MC Sim	4.07	12.79	19.28	5.40
Reco 1		6.97		
Reco 2	1.42	7.06	11.38	5.22
Reco 3	1.27	7.49	5.25	21.50
Reco 4	9.04	7.44	4.85	8.68
Reco 5	2.56	13.93	8.95	4.74
Reco 6	1.29	7.98	10.75	13.34
Reco 7	4.08	10.13	10.72	11.85
Digi Reco 1	5.48	3.94	18.20	4.47
Digi Reco 2	9.66	7.63	16.19	3.55

Table 7.10: Comparing the Clouds. Displayed are the standard deviations of the wall time, after splitting the VMs of T-Systems and IBM into two categories. The fast category contains two VMs for T-Systems and three VMs for IBM, excluding event generation which was not impacted by the different speeds.

T-Systems	Wall Time [s]	CPU Time [s]	Idle Time [s]	I/O Wait Time [s]
Digi Reco 3	14552 ± 1091	17429 ± 5501	97190 ± 8370	93 ± 13
Digi Reco 4	109193 ± 1276	81535 ± 1235	784891 ± 8975	182 ± 9
Digi Reco 5	109897 ± 1146	81094 ± 1529	790626 ± 7596	223 ± 8

Table 7.11: In this table, the digitisation test results of the jobs run on the T-Systems infrastructure are shown. Jobs that took less than 500 seconds CPU time are excluded, as these failed.

provide an object storage and all jobs using the Exoscale object storage failed. They were doing direct I/O, accessing the data remotely.

Digi Reco 3-5 represent the attempt to evaluate the workflow performance with data input from the T-Systems object storage. Digi Reco was chosen, because it is the workflow that has the biggest input file size. Furthermore, Digi Reco is the only workflow that is reusing input data between different jobs. Therefore, this workflow would be the most logical choice to make use of native Cloud storage.

Due to time constraints and frequent failures, the two last digitisation measurements (Digi Reco 4 and 5) contain only a few (18 and 9) measurements, which are evenly distributed amongst the VMs. Digi Reco 3 fetches its input data via XRootD remotely from EOS, which is located at CERN. In contrast to that, Digi Reco 4 and 5 fetch it from the T-Systems object storage. Digi Reco 5 is a repetition of Digi Reco 4. The bucket within the object storage was accessed via the https interface that T-Systems provides for files within buckets. The job details for Digi Reco 3, 4 and 5 can be found in the Appendix, see Subsections A.6.12, A.6.13 and A.6.14.

Another problem that occurred, was that constraints with the utilised AthenaMP version made it impossible to run Digi Reco 3-5 with AthenaMP. They were therefore run in single-core mode and cannot be compared easily to the previous Digi Reco 1 and 2 workflows.

What can be seen is that the results of Digi Reco 4 and 5 are similar, which is unsurprising as they represent the same jobs. Comparing these to Digi Reco 3 shows that it is much faster to have direct I/O access to the input data through EOS than the object storage. Considering that EOS was located in a different data centre, further away, this is unexpected.

As a cross-check, the input data was copied to the local disk from the two different sources. The download speed revealed that downloading the input dataset from EOS to the local disk takes about 20 minutes. Downloading it via the S3 API from the object storage takes around 16 to 18 minutes. This is the opposite behaviour from what was seen for the job duration using direct I/O. This means, that the bandwidth does not explain the large discrepancy and that the direct download slightly favours the object storage.

This was investigated further, even after the T-Systems exercise was finished. One possible explanation for the inflated numbers of Digi Reco 4 and 5 is the different access mechanism, which happened via https. The “s” in https stands for secure and it has been shown, that it can have a negative impact on the performance compared to http. In order to reject this theory, the job was repeated with input data from CERNBox that was exposed via https as well. A significant slow down was observed there as well, so the theory could not be rejected. As there was no possibility for further tests on the T-Systems infrastructure, it will remain uncertain whether the object storage contributed to the slow down.

The high idle time is a result of the high network activity.

7.2.3 Grid sites

In order to make a comparison, below are the numbers of how the jobs performed within the WLCG. The two Grid sites chosen for comparison are CERN (CERN-PROD-preprod_MCORE), located in Geneva, and GoeGrid (GoeGrid_MCORE) in Göttingen. Both queues consist of 8-core VMs. Also, the single-core queue of Göttingen is used for a comparison of the event generation jobs. The usage of queues already indicates, that the jobs run within the Grid were regular production jobs, run through PanDA. There was no special treatment or customisation, which is why they differ in the number of processed events, compared to what was run within the Cloud. In addition, not all jobs that ran within the Cloud corresponded to production jobs, meaning only a subset of jobs is compared.

Furthermore, the monitoring information that can be extracted from the PanDA monitoring is limited to the Wall and CPU times.

CERN	**Wall Time [s]**	**CPU Time [s]**
MC Sim	**28952 ± 10821**	222503 ± 82196
Reco 5	**3470 ± 1163**	10660 ± 2606

Table 7.12: In this table, successful Grid jobs run at CERN that correspond to the tests run within the Cloud are displayed. The Reco 5 jobs at CERN processed only 2025 events, whereas the ones within the Cloud providers processed double that number. The Reco 5 numbers are taken from 69 jobs. MC Sim processed 1000 events on the Grid and 100 events within the HNSciCloud exercise.

Göttingen	**Wall Time [s]**	**CPU Time [s]**
EvGen	**1895 ± 257**	1774 ± 240
MC Sim	**14090 ± 4361**	109106 ± 34246

Table 7.13: In this table successful Grid jobs run at Göttingen that correspond to the tests run within the Cloud are displayed. The EvGen jobs were run on single-core VMs within the GoeGrid queue. EvGen processed 500 events on the Grid, whereas it processed 1000 events within the Cloud. MC Sim processed 1000 events and 100 events within the HNSciCloud exercise.

What can be seen in the two Tables 7.12 and 7.13 is that the fluctuations within similar jobs are high, both for CERN and for Göttingen. For MC Sim and Reco 5 they are above 30%. Only the single-core EvGen workflow has a lower fluctuation of around 14%. This is unsurprising, as it is known that there are several different generations of hardware within both computing centres. It has also already been observed when comparing the Cloud measurements, that EvGen seems to be less affected by different

types of hardware.

A direct comparison of the absolute numbers is possible between the two Grid sites, which shows that the performance of GoeGrid is much higher compared to CERN. This can be gleaned from the direct comparison of the two sites MC Sim numbers. The absolute performance between the Grid and Cloud is more difficult to compare, as the jobs processed a different number of events. A discussion about comparing jobs with a different number of events is briefly done in Subsection 7.3.4. A rough estimation can be gained by, for example, doubling the wall time of Reco 5 and comparing it to the Cloud performances. Following this, CERN performs roughly between T-Systems and IBM. For MC Sim, the event discrepancy is a factor of 10. Multiplying the Cloud wall times by this factor leads to an inflation, as the overheads are basically included ten times, instead of once like for the Grid numbers. The performance of CERN would still be placed in between T-Systems and IBM.

Comparing the GoeGrid performance to that of the Cloud providers, places it between Exoscale and IBM.

7.3 Model application

The above measurements show that the largest fluctuations originate from differences in the infrastructure, rather than variations in the workflows. This opens up the question of, whether it is necessary for the WIM to benchmark on more than one VM. Increasing the number of VMs will increase the accuracy, but the question is: how many VMs should be benchmarked? If the subset of the VMs does not represent the final composition, it is not very useful. In order to get the most accurate prediction, it would be ideal to benchmark all of the VMs that will be used. Additionally, as was shown above, the performance of a VM can change over time, so regular benchmarks would be required.

Following these considerations, the cost that is incurred from benchmarking is very high. For Cloud computing, the duration of the benchmarks can be directly translated to a cost. On the Grid, every second a benchmark is run, no data processing is taking place.

For the WIM application as little benchmarking as possible was used. Since the benchmarking is done with actual workflows, no resources are wasted, see passive benchmarking in Subsection 4.6.1. As it turned out, for the provider with the largest variations, two different generations of VMs were used, which were modelled separately. In that case, looking at more than one VM was useful, although a larger amount of VMs is needed to have a relative certainty that all variations are included. It is necessary to know however, that the VM is not an outlier.

In general, the VM performance should be guaranteed in the contract with the Cloud provider. Therefore deviations from that contract should only be towards higher performance. When using only one VM for the benchmarking, it needs to be clear that the VM is not an outlier. The WIM then estimates the lowest boundary and the average should be higher.

In addition to this section, another WIM application can be found in Chapter 8.

7.3.1 Combining benchmarks

The idea behind combining benchmarks is that not every workflow needs to benchmark the infrastructure anew. This approach was already used to a certain extent, as *similar jobs* are grouped together and benchmarked only once. The question that remains is: to what extent can workflows be grouped together and still produce accurate WIM results?

Using the benchmark results of one workflow to describe the infrastructure for a second workflow is from here on called cross-modelling.

Table 7.14 shows the cross-modelling results of three different reconstruction workflows on single VMs. The reason why single VMs are used, is to keep the base line of fluctuations low, in order to detect smaller model discrepancies.

Model discrepancies [%]	**IBM VM1** Wall Time	**T-Systems VM1** Wall Time	**Exoscale VM1** Wall Time
Reco 1	1.11	-0.80	-0.91
Reco 2	-3.74	1.64	0.97
Reco 3	0.48	-1.53	-0.52
Reco 1 with 2	**31.96**	**-13.96**	**13.90**
Reco 1 with 3	7.41	**-19.46**	3.04
Reco 2 with 1	**-23.22**	6.60	**-13.59**
Reco 2 with 3	**-18.73**	-6.63	**-10.16**
Reco 3 with 2	**18.56**	5.94	**11.22**

Table 7.14: In this table, model deviations from the measurements are shown. The top three rows are the regular scenario that has been shown before. In the other rows, cross-modelling is taking place. The benchmark results from one reconstruction were used in order to describe the infrastructure of another reconstruction. For example, Reco 1 with 2 means that Reco 1 was modelled with the infrastructure input from Reco 2. Percentages larger than 10% are highlighted in bold.

What can be seen in the table is that the model discrepancies are below 4% for the single VM and typical benchmark cases. When cross-modelling, the discrepancies increase drastically to over 10% for most cases, reaching up to 32% for Reco 1 with 2 at IBM. Some infrastructure-workflow combinations work better than others, but there is no clear pattern that can be discerned.

When looking into the wall time components of the cross-modelling, the biggest CPU discrepancy is 4.53%, staying below 3% in most cases. The I/O wait time is in general at least a factor of ten smaller than the CPU or idle time, so the impact of a wrong prediction on the wall time is relatively small. The large wall time discrepancies therefore result from the idle time.

7.3.2 Bandwidth estimation

This use case was already presented in the CHEP paper [117]. It is one of the reasons the WIM was created in the first place. The issue that came up was to determine how much bandwidth would be required by running ATLAS reconstruction on a given Cloud infrastructure. The Cloud site was T-Systems, but the activity happened before the HNSciCloud activity. There was a valid concern that the existing 10 Gb/s WAN link between this Cloud site and CERN would not be sufficient. The link would have to satisfy the input requirement of 4000 CPU cores running reconstruction in parallel. In order to evaluate this concern, the WIM was used.

It highlights an example in which a typical infrastructure input value can also be investigated as an output. The result showed that the existing network would be enough, regarding the specifications: 1000 x 4-core VMs, 116 HS*s/evt CPU time, 850 kB/evt input, 2701 Evts per job and 1417 kB/s instantaneous network read. The caveat was, that the exact specifications of the infrastructure were not clear, one example being the computing power. Back then, the benchmarking approach did not yet exist. The solution to accommodate different types of hardware, was to model a broad range of e.g. the computing power. This means, that the computing power became a varied input and the WIM calculated results for many different ones. In order to be thorough, changes within the ATLAS job configuration that were known to happen, were also considered. These parameters were then varied. The resulting bandwidth gave indications of what is to be expected, as well as worst case scenario estimations. Figure 7.6 shows the result.

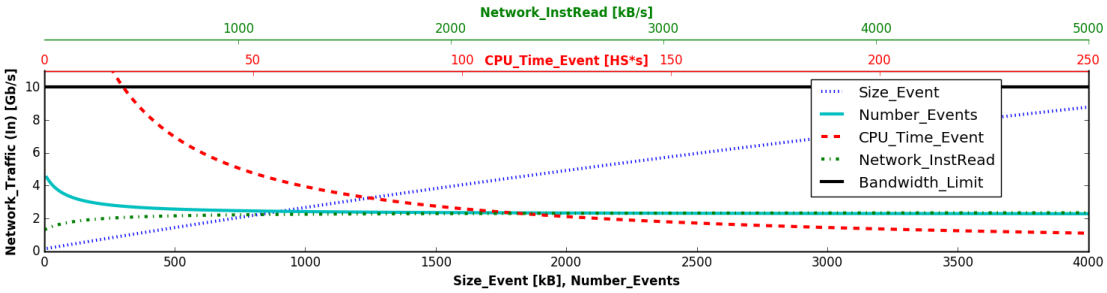

Figure 7.6: Infrastructure limit and Model output [117].

The 10 Gb/s bandwidth limit (Bandwidth_Limit) is depicted by the black horizontal line. Any bandwidth estimations going over this line indicate a possible bottleneck, if the underlying conditions are met.

Within the chosen range, the impact of the size of the events (Size_Event), the number of events per job (Number_Events) and the instantaneous bandwidth (Network_InstRead) is small enough not to be of concern. The instantaneous bandwidth is relevant when reading the input data event by event.

The one variable that could make the result reach the critical value is the CPU time

per event (CPU_Time_Event). Taking a closer look, it can be seen that the cross over point is reached at ~ 20 HS $*$ s. For the processing this would mean either a decrease in event complexity, which is unlikely as the complexity is rising with the pileup or a drastic increase in CPU power. Extrapolating from the technological evolution of the past several years leads to the conclusion that the gain in computing power over a year is small and this is unlikely to happen.

7.3.3 HNSciCloud large scale

Previously, the results of the measurements on the HNSciCloud were discussed. The WIM was used to predict the performance of the different Cloud providers.

As the resulting performance was already discussed, Table 7.15 shows the deviation of the Model results from the measurements.

Model Discrepancies [%]	**Exoscale**	**IBM**	**IBM fast**	**T-Systems**	**T-Systems fast**
EvGen	1.92	0.62		2.15	
MC Sim	4.47	6.39	3.07	-36.01	21.80
Reco 1	-4.46	-0.87	-1.38		
Reco 2	-3.97	-2.09	0.83	-18.80	7.71
Reco 3	-6.01	1.73	-2.26	-3.05	-5.70
Reco 4	2.39	0.73	3.03	3.41	2.00
Reco 5	-3.97	-2.09	0.83	-18.80	7.71
Reco 6	1.25	1.09	-1.30	-19.02	10.88
Reco 7	8.04	-5.08	8.72	-23.24	5.77
Digi Reco 1	-10.55	1.72	8.24	-21.77	14.18
Digi Reco 2	0.80	3.43	11.92	-4.77	14.23

Table 7.15: Model error on the large scale HNSciCloud tests using ten VMs per provider.

The model predictions for Exoscale and IBM are deviating from the actual performance by only a small margin. The biggest discrepancies, exceeding the 20% goal, can be found with T-Systems, even though it is already divided into the two VM generations. The explanation is that the VM on which the benchmarks were performed, deviated in its performance from the average. In addition, the workflow performance within a VM fluctuated more than in the other providers. This is consistent with the higher standard deviation that was found for T-Systems compared with the other providers.

7.3.4 Error sources

Idle time measurement

One of the reasons for the TSY model discrepancy comes from the idle time. The setup of the T-Systems VMs differs from the others. It seems to be slow with regards to I/O

operations, which manifests itself in a low IO-factor. In this particular case there seems to be a correlation between the IO and the idle time, which is shown below.

In Figures 7.7 and 7.8 the monitoring numbers of the T-Systems VM and the Exoscale VM are plotted. Comparing the idle profile of the two figures, which can be found on the bottom plot, looking at the pink curve, indicates a difference in the overall profile. This is especially apparent between 16:55 and 17:25 in the T-Systems plot. This period is during a CPU intensive stage in the job.

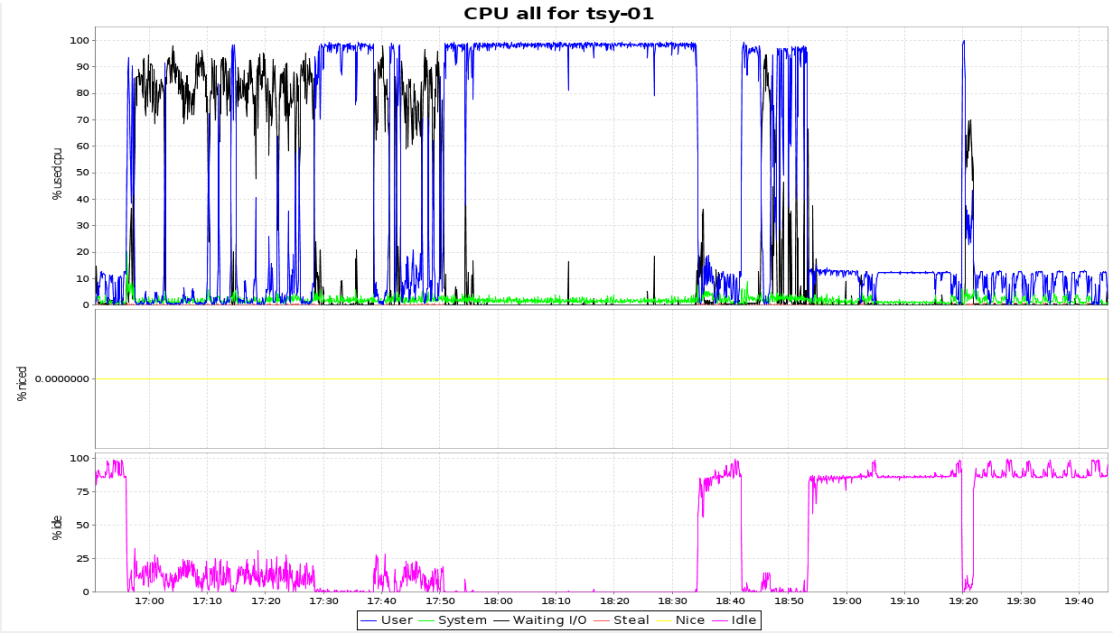

Figure 7.7: T-Systems VM CPU sysstat monitoring result of a reconstruction job with merging at the end.

In Figure 7.8 the CPUs are either busy or waiting for IO in that phase. For T-Systems, there seems to be a percentage of CPUs in idle state, in parallel with a high IO wait, which can be found in Figure 7.7, looking at the black curve of the top plot. The high IO wait is not consistent throughout the job, therefore, for example, between 18:00 and 18:30 no idle time was measured. A change in the amount of idle time that happens in parallel to the heavy IO-wait period, will have an impact on the idle factor.

One reason why the T-Systems VMs may behave differently is the different setup, such as the storage that is attached through the network. Influences from other users may lead to congestion and therefore this CPU inefficient job behaviour.

Idle factor

Due to the job characteristics, the initial assumption for jobs that have very little network activity was that the idle time of the target machine can be obtained from the reference

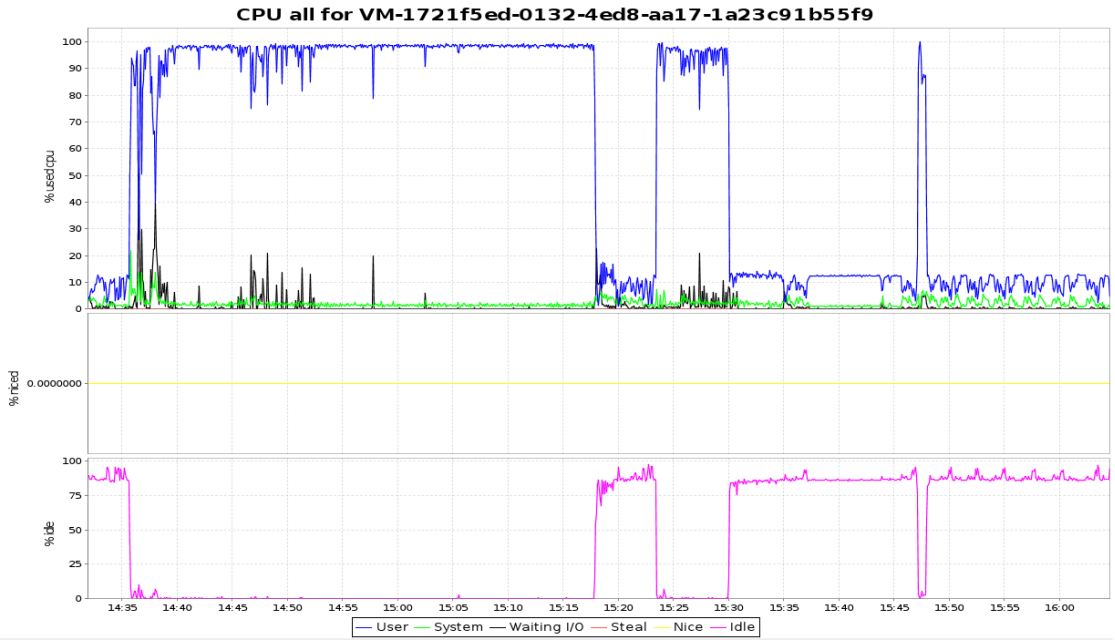

Figure 7.8: For comparison: Exoscale VM CPU sysstat monitoring result of a reconstruction job with merging at the end.

machine, through the following equation:

$$Idle\_time_{tg} = \frac{Idle\_Time_{rf} * CPU\_power_{rf} * Nr\_Evts}{CPU\_power_{tg}} \quad (7.1)$$

The logic being, that the faster the CPU the lower the idle time. This is because during a serialised phase such as merging, the duration of the serialisation depends on how fast the merging is performed using only one core at a time. If the serial part that is waited upon finishes faster, the idle time is decreased. Depending on how many CPU cores the VM has, the decrease can be significant. This assumption held for event generation and Monte-Carlo simulation jobs, for which the idle time is low, but not for reconstruction jobs (see Chapter 7). In order to gain a more accurate result, the idle factor was introduced.

The idle factor greatly increased the accuracy of the model, when the benchmark was similar to the workflow. In the case of modelling, for example, a reconstruction job with little merging, but using input values from a job that heavily merges, the results using an idle factor were worse, see Table 7.14, than the ones without, see Table 7.16.

This behaviour seems surprising, considering that the infrastructure itself is the same for both reconstruction 2 and 3. It can be explained by a change in the overall workflow pattern. The disk access, for example, happens in a completely different manner than before, inflating or deflating the disk speed. An explanation lies in the different access

patterns, which are associated with different disk performances, such as random and sequential read speed.

In addition, the results for the I/O wait time are much worse, but the absolute I/O wait time is much smaller than the idle time, making its impact on the wall time small.

In Table 7.16 the cross-modelling discrepancies without an idle factor input are shown.

Model discrepancies [%]	**IBM VM1** Wall Time	**T-Systems VM1** Wall Time	**Exoscale VM1** Wall Time
Reco 1 with 2	**17.68**	**-18.63**	3.18
Reco 1 with 3	1.41	**-18.78**	0.58
Reco 2 with 1	**-25.25**	1.94	**-12.48**
Reco 2 with 3	**-23.22**	-5.80	**-12.26**
Reco 3 with 1	-8.71	7.83	-4.35
Reco 3 with 2	3.41	-0.83	-0.54

Table 7.16: Modelling across workflows without an Idle factor.

Comparing these results with the previous results, that had an idle factor as input, see Table 7.14, shows that the model deviations have mostly become smaller. The results are still not satisfactory. They show, however, that in cross-modelling it is better to estimate the idle factor from the CPU power rather than from the performance of a different workflow. This is not the case for regular modelling, where the benchmarked idle factor results are more accurate. Since there is no additional benchmarking needed to retrieve the idle factor and since the workflows are carefully classified, the approach that uses an idle factor as input was chosen as the WIM default.

Number of Cores

After several measurements on single-, 4- and 8-core VMs, it became clear that the number of cores is an additional potential error source. The error gets introduced when benchmarking a VM with a certain number of cores and then trying to model an infrastructure that consists of VMs with a different number of cores. It can be easily avoided by only comparing VMs with the same number of cores.

It was found that modelling, for example, MC simulation with event generation infrastructure input parameters gives large discrepancies. This seems to be surprising, as both workflows are CPU intensive and do not include many I/O operations. The explanation is the single core nature of the event generation job.

Using single-core CPU factors to predict multi-core CPU factors introduces a large error, even if the jobs are similar. This is in part due to overheads that are introduced by running on multiple cores. One example would be that all AthenaMP workflows process the first event on only a single core. For reconstruction, a large discrepancy between the CPU factors stems from the serial merging step, that is not performed in the single core scenario. Additionally, the CPUs and operating systems themselves use different

optimisations that have a different impact on the single- or multi-core scenario.

Number of events

It is also possible to extrapolate the number of events that are processed per workflow between the benchmark and the actual workflow. This is not recommended, as the overheads are not considered separately in the latest version of the model. The overheads will therefore have a higher impact in a job with fewer events. This means that when benchmarking, for example, a workflow with five times fewer events than are to be modelled, the resulting wall time will be too large by four times the overhead. It can be avoided by benchmarking the workflows with the correct length.

This number of event extrapolation was however used in Subsection 7.2.3 for comparing the different results. The difference in the overheads was taken into consideration and the overall wall times were only compared very roughly.

CHAPTER 8

Overcommitting

"The perils of overwork are slight compared with the dangers of inactivity." - Thomas A. Edison

8.1 Principle

Overcommitting means allocating more resources than are available. In computing, this means hosting virtual workflows that have a higher maximal demand than the physical availability [126]. This can be in terms of all components, CPU, Network, Storage or RAM. The idea behind overcommittment (OC) is that it counteracts underused resources.

This can happen in several ways. In Cloud computing, VMs (users) located on a physical machine are maximised, distributing a maximum workload that exceeds the physical capacity of the physical machines [127]. Cloud providers use this technique to exploit the effects of resource overprovisioning by customers, which can be significant.

One example is the Cloud computing cluster data released by Google. Google allocated 80% of the actual memory and over 100% of the actual CPU capacity of its cluster. Still, the overall usage was only at 50% of the total capacity for memory and 60% for CPU [128]. This difference between resource requests and actual usage, is a pattern that can be observed throughout the whole IaaS business. This can also be seen, for example, in papers by Liu, Ghosh and Naik and Dabbagh et al. [129] [36] [127]. In these examples, the actual resource utilisation is even lower than described above.

There are several reasons for these wasted resources. Due to the Bullwhip Effect [130], it is often hard for a customer to know the exact amount of resources that are needed,

leading to a VM sprawl [131]. The resource consumption of a VM can vary with time, meaning it is often not running at maximum capacity. An example would be a web server, that is accessed more or less frequently during the course of a day [127]. The same that is true for one VM can equally apply to a customer that uses several VMs. In general, Cloud providers are using some sort of OC, however they do not disclose how, or how much, of their resources are overcommitted.

A second way in which OC can be beneficial, is underutilised CPUs due to the workflow profile. Jobs at CERN, for example, are running around the clock and almost all of the available resources are occupied by jobs full time.

However, the CPUs are not utilised 100%, leading to another kind of inefficiency. As seen before, a CPU that is computing a job can be in an idle state or waiting for input data to compute, known as the I/O wait time. Depending on the job type, the CPU efficiency (see Subsection 4.6.4) varies greatly. Simulation jobs have a very high efficiency, greater than 95%, while multicore digitisation and reconstruction (DigiReco) jobs in particular have a low efficiency of around 60%. These DigiReco jobs read a large amount of input data. The example jobs used in Subsection 5.3.4, for example, read 32 GB per 1000 events and the CPU is often waiting for input.

Hyperthreading is one example of OC that is used on several WLCG sites [132]. It creates two virtual processors for every logical processor. Overcommitting is thus introduced at the infrastructure level. Some workflows, such as DigiReco, profit more from hyperthreading than others, such as simulation which does not benefit from it at all [133].

A low job-efficiency can be observed throughout the WLCG, this is due to the job mix described in Chapter 5. A large portion of jobs is I/O intensive. The idea on how to improve the CPU utilisation is to use OC in order to occupy idle CPU cycles during the time they are idle or in I/O wait. In contrast to the previously explained OC use cases at the infrastructure level, this OC happens at the job level. This means that the infrastructure setup is not changed, but the jobs themselves run more parallel processes than there are (virtual) CPU cores.

The biggest difference between the two levels of OC is the knowledge about the resource usage. For example, a Cloud provider does not know how much resources a customer will consume. Therefore, the provider overcommitts a percentage of its resources, including a safety margin, that is based on their best guess. For OC at the job level, the profiles of the workflows are well known and can be adapted to each other by choosing the best combinations. The obvious choice would be to combine a very I/O intensive job with a CPU bound job, or vice versa. In the first case, the OC makes use of the fact that disk I/O is a bottleneck and CPU resources are free, while at the same time not piling additional I/O on top.

An example would be the workflow profile presented in Subsection 5.3.2, which indicates that there is room for optimisation, especially during the single-core merging steps.

There are also downsides to OC. The biggest drawback is that more workload on a

VM means that it will require more memory. This case is similar to the case, in which more logical CPU cores than physical cores are created.

Secondly, the job submitting frameworks of the experiments are very inflexible. In the ATLAS case, the attempt to test OC on the Grid failed due to various restrictions. Circumventing the standard user submission and attempting to overcommit via the HammerCloud functional and stress testing framework [134] [135] was also unsuccessful. Therefore, it is difficult to implement OC at the job level.

If there are too many parallel processes running at some point they might suffer from resource contention. This means that the processes are competing for resources amongst each other, slowing down the VM.

8.1.1 Overcommitting scenarios

In Figures 8.1 and 8.2, a theoretical look at the progression of overcommitted multicore applications is sketched. It highlights how the OC works and from where the possible gains will come, making it clear which of the different scenarios can benefit more or less from OC. For simplicity, the example in these Figures does not include memory or other I/O considerations.

In Figure 8.1, the overall duration increases from 32 s to 40 s, whereas in Figure 8.2 it only increases from 32 s to 34 s. The only difference between these two scenarios are the types of workflows that are run. In the first example, no CPU idle or I/O wait time is prevalent. Therefore the five processes cannot be parallelised further and the overall duration is the sum of the duration of all workloads divided by the number of cores. The second example shows how this result can change if the workflows have a lowered CPU efficiency. For example, data-intensive workflows with an increased number of I/O wait periods provide the potential for more workload to be done.

In Figure 8.1, the event yield is exactly the same with or without OC. This can be seen, when interpreting the figure in the following way: one second of CPU activity of Process 5 (red box) represents one event. Since Process 5 (red) and Processes 1-4 (blue) are equivalent, one second of Processes 1-4 (blue) also corresponds to one event. The Non-OC and OC yields are therefore equal:

$$non\_OC\_yield = \frac{128}{32}\frac{evts}{s} = 4\frac{evts}{s} \equiv OC\_yield = \frac{160}{40}\frac{evts}{s} = 4\frac{evts}{s} \quad (8.1)$$

evts stands for events.

The scenario and yield change when looking at Figure 8.2. Here, CPU idle and I/O wait times (white boxes) existed and could be replaced by Process 5 (red boxes). This means additional workload was done in the OC case and therefore the yield increases:

$$non\_OC\_yield = \frac{92}{32}\frac{evts}{s} \approx 2.9\frac{evts}{s} < OC\_yield = \frac{124}{34}\frac{evts}{s} \approx 3.6\frac{evts}{s} \quad (8.2)$$

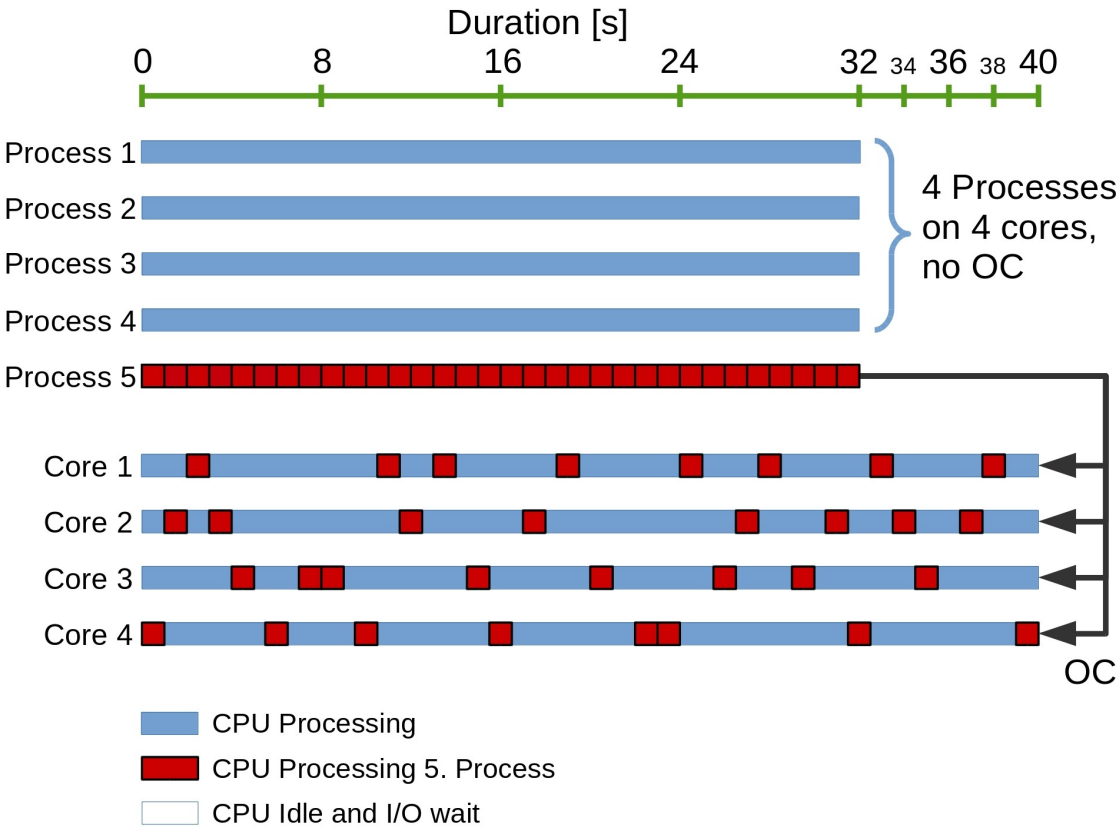

Figure 8.1: Schematic view of the OC progression. The green scale at the top represents the elapsed time in arbitrary units. Underneath are four processes, indicated by four blue bars. These bars indicate the usual progression of these four processes on a four core machine (non-OC scenario). After adding another process (indicated in red), the picture changes (following the black arrows) to the four bars labelled "Core 1-4". The fifth process is equivalent to the other four, the red colour is to highlight the differences between the two scenarios. This is a schematic view of how the five processes would run on a four core machine (overcommitted). Since the processes are similar, the scheduler distributes them equally amongst the CPUs.

The distribution of the processes, and the interactions between them and the CPUs, are much more complex than shown in Figure 8.2. In reality, each of the blocks from the example would last only a few milliseconds and be switching between the CPU cores much more. Furthermore, the fifth process would include some CPU idle and I/O wait blocks.

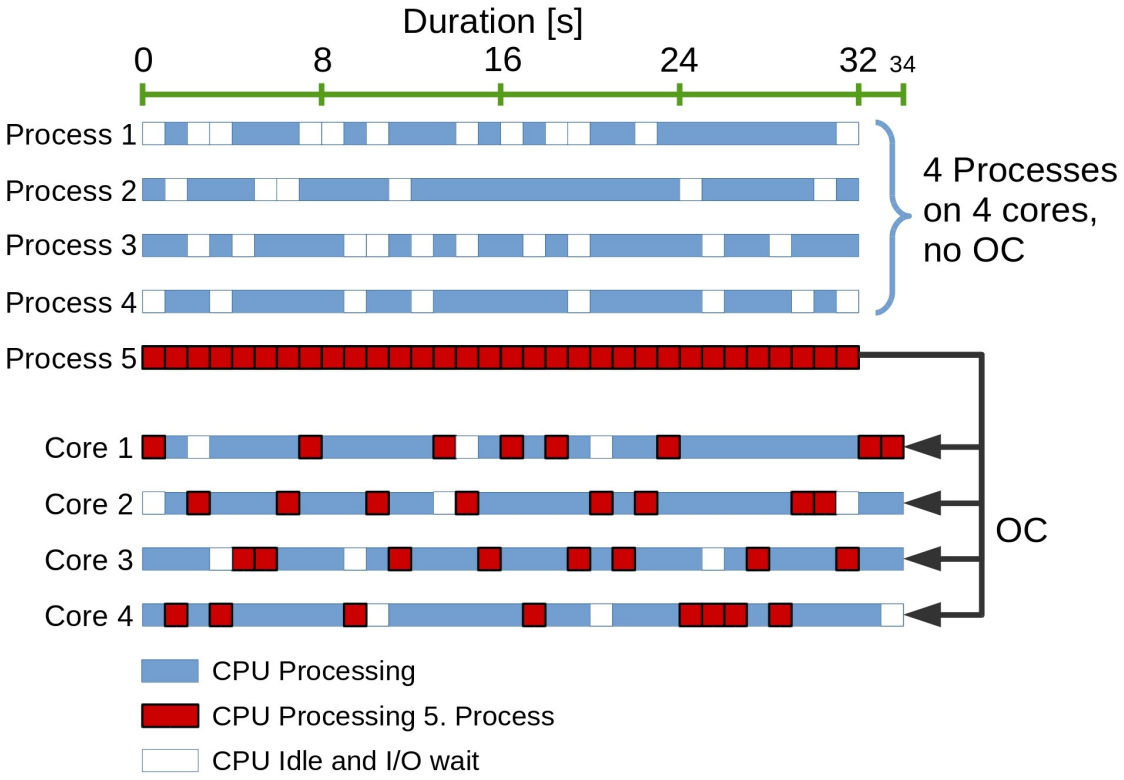

Figure 8.2: Schematic view of the OC progression. The green scale at the top represents the elapsed time in arbitrary units. Underneath are four processes, indicated by four blue bars. These bars indicate the usual progression of these four processes on a four core machine (non-OC scenario). The white boxes indicate idle or I/O wait times. After adding another process (indicated in red), the picture changes (following the black arrows) to the four bars labelled "Core 1-4". The fifth process is equivalent to the other four, the red colour is to highlight the differences between the two scenarios. This is a schematic view of how the five processes would run on a four core machine (overcommitted). Since the processes are similar, the scheduler distributes them equally amongst the CPUs.

Overcommit factor

In order to quantify the benefits of OC, the overcommit factor "OCF" is introduced. Unused CPU cycles of the non-OC case can be potentially used by OC. In reality, not all of this potential is actually used. The OCF gives the percentage of the OC workflows that were computed using the unused cycles of the non-OC scenario.

This can be illustrated with the help of two examples. In Figure 8.1, there is no CPU

idle or I/O wait time in the non-OC case, indicated by white boxes. The additional yield, see Subsection 8.1.1 and the OCF are zero, $OCF = \frac{0}{32} = 0$. The fraction represents the additional workload (red boxes) computed instead of the idle and I/O wait time (white boxes) divided by the total workload.

In contrast to the first example, there are unused CPU cycles in the second example: in Figure 8.2 the OC used potential CPU cycles that were available in the non-OC scenario. Comparing the non-OC and OC scenarios, some idle and I/O wait times (white boxes) have disappeared, replaced by Process 5 processing (red boxes). The yield in the OC case is therefore increased. The OCF is $OCF = \frac{24}{32} = 0.75 \equiv 75\%$. The numerator of 24 corresponds to the number of red blocks that were computed instead of previously white blocks.

According to this definition: $0 \leq OCF \leq 1$. The worst case, with no OC benefits and $OCF = 0$ has already been shown in Figure 8.1. The best case of $OCF = 1$ would be achieved if the OC scenario has the same wall time as the non-OC scenario. In Figure 8.2, an $OCF = 1$ would mean the OC wall time is (32 s). This can only be the case if all Process 5 processing (red blocks) replaces previously unused CPU cycles (white blocks).

The next Section 8.2 shows the OC results of real ATLAS workflows.

8.2 Study

8.2.1 Overcommitting with AthenaMP

The simplest way to overcommit is to make use of the built-in AthenaMP functionality that determines the number of parallel processes that are spawned. Changing the number of parallel processes does not change the workload, meaning the number of events that are processed stays the same. In Figure 8.3, the different wall times resulting from a varied number of parallel processes are shown. Each bar represents the average over three jobs, the error bars depict the standard deviation. The variation of the wall time amongst the sets of three jobs was very low. The best configuration is at eight processes, as it takes the shortest amount of time for the same workload. This is expected, if one considers the job profile, one of which is shown in Figure 5.9.

The profile regions in which the setup would benefit the most from additional processes are the serialised phases, i.e. the merging steps. However, these regions do not see an increase in CPU utilisation. This is due to the AthenaMP restrictions that produce the same outcome, meaning that still only one CPU core is used when merging. Therefore, the nine and ten core scenarios do not benefit from OC but suffer due to the increased memory requirement and the resource contention.

The effects of too little memory have been shown in Subsection 5.3.2. In this case, OC will only make the situation worse. This is due to the fact, that the memory requirement

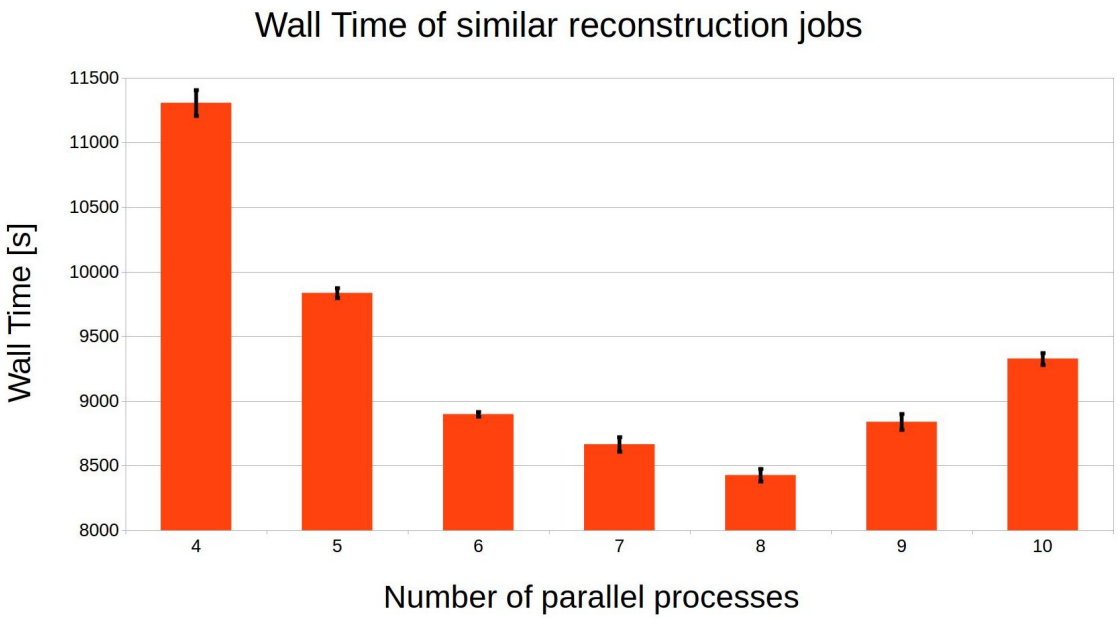

Figure 8.3: Raw data reconstruction job duration averages, combining three repetitions, including the standard deviation (see Equation 5.1), of an increasing number of parallel processes on an 8-core VM. Note: for better visibility, the y-axis does not start at zero. The exact numbers can be found in the appendix, see Table A1. The input data was on the local disk.

goes up with each additional process. The effect of increasing the number of processes is therefore similar to lowering the amount of available RAM, in the sense that the VM will have to start swapping in both cases. Therefore, it is important to know the memory requirements and availabilities. Only then can the decision be made, whether OC will deliver additional performance, or the opposite.

The reason why the undercommitted case takes longer, is that a part of the VM is not used for processing, lowering the CPU efficiency further and increasing the duration.

Since this kind of OC does not produce any benefits, it is not used in the following. The next approach that is used for the rest of this chapter, was to simultaneously start multiple AthenaMP instances, instead of just one. In that scenario, OC is achieved from the combined number of parallel processes of the two or more AthenaMP instances, even if each individual instance does not overcommit.

8.2.2 Overcommitting job profiles

In Chapter 5, the profiles of the different workflows were examined. This subsection will show the changed situation when OC is taking place. The following examples were executed on the same VM, located at CERN, for more details see Subsection A.7.2 in

the appendix. The time-axis of the following graphs has the same range, so the wall time between different jobs can be compared more easily. The same applies to the different y-axes.

In order to limit the number of profiles displayed in this chapter and to avoid confusion, only the two most significant Figures 8.4 and 8.5 are displayed. Together they combine all the information and changes. Additional graphs that highlight the changes step-by-step can be found in the appendix.

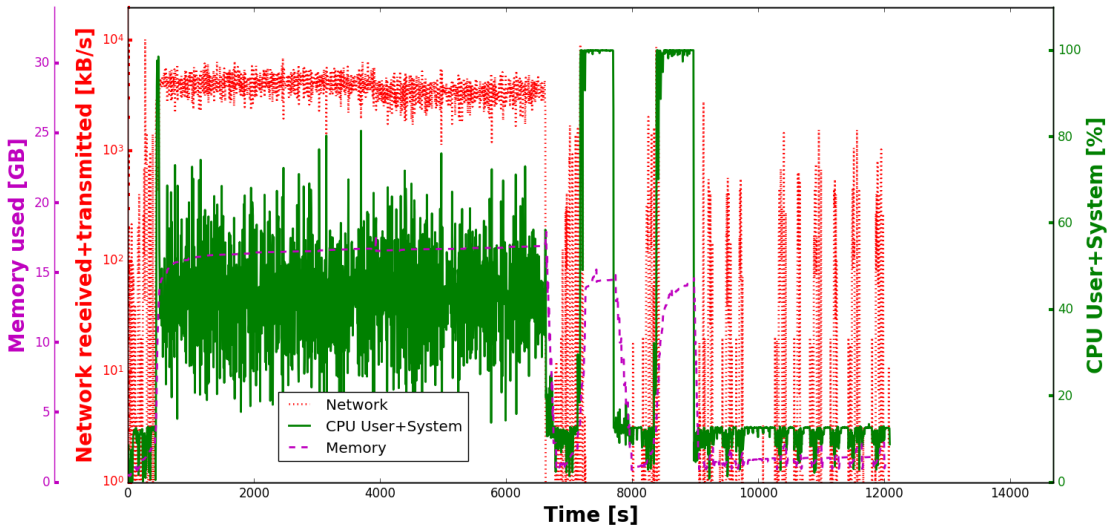

Figure 8.4: Reconstruction profile, showing the execution of eight parallel AthenaMP processes on an 8-core VM. The input data was not on the local disk. It was read through the network from a remote storage at BNL across the Atlantic.

In Figure 8.4 a reconstruction profile of a job reading the input data from a remote location, is shown. The remote reading is used by CMS and much effort has been put into investigating it by ATLAS. The usual setup of eight parallel processes on eight cores was used. It can be seen that there is a high network activity during the first large processing step (RAWtoESD, 0 s to around 6500 s). At the same time the CPU usage is below 100%, it stays below 60% most of the time. This is in contrast to the case of local data that can be seen in the appendix in Figure A9. There the network activity is low and the CPU usage high.

The large contrast results from the fact that the remote data location is purposely chosen to be far away at BNL, so a higher latency comes into effect. The "ping" command between the VM and a ping-able host at BNL shows an average latency over 10 packets of 93.775 ms with a standard deviation of 0.221 ms. The latency manifests in the lower CPU usage (from 100% down to 30-60%), as the CPUs are constantly waiting for input data to be read from the remote storage. For an even more granular comparison, in the Appendix, see Figure A10, a profile of the same job with remote data input, but a low

latency, can be found. There, the negative CPU usage impact is much less.

What can also be noted is the merging step at the end after around 9000 s. The merging is done serially, meaning only a single CPU core is used. The disk read/write limit is not reached during the merging and most of the merging step is CPU bound.

The profile changes significantly with OC. In Figure 8.5, the number of processes was doubled, by executing the same reconstruction twice at the same time. In contrast to before, the total workload was also doubled. This increased the number of parallel processes to 16. The choice of doubling the processes and workload is a consequence of an AthenaMP restriction that is discussed in Subsection 8.3.1.

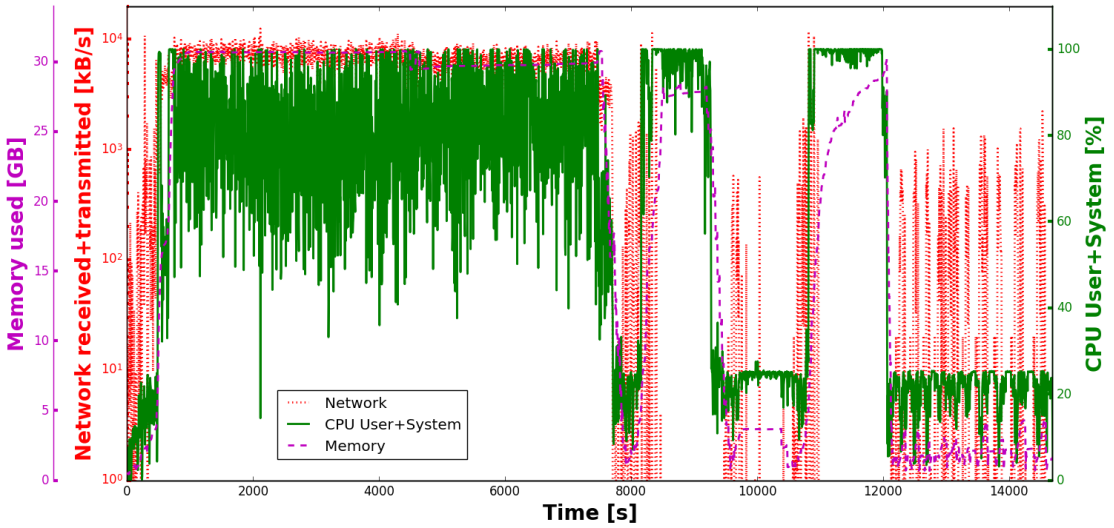

Figure 8.5: Overcommitted profile, showing the execution of two times 8 (16) parallel AthenaMP processes on the same 8-core VM. The workload is doubled. The input data was read through the network from a remote storage at BNL across the Atlantic.

The two jobs together did not require double the amount of time, even though they were computing twice the workload, see Figure 8.5. The profile shows, that during the serialisation, the CPU utilisation is doubled. This gain in the CPU utilisation in the serialised part of the job is prevalent in the other scenarios of local data or low latency remote data input as well. In the Appendix, see Figure A11, a profile of the same overcommitted job with remote data input, but a low latency can be found. It can be seen, that the memory requirement is much higher for all OC cases. It is more than 30 GB, compared to around 16 GB from before, see purple dashed lines.

There is an additional increase in the CPU usage between the non-OC and OC scenarios in Figures 8.4 and 8.5. It can be seen during the first processing step (RAWtoESD,

0 s to around 6500 s and 7500 s) and it raises the CPU usage from 30-60% to 60-100%. It originates from the fact that, while some processes are waiting for input, others can be computed. The high latency to BNL guarantees that there is a significant percentage of CPU inactivity. The additional processes can make use of these inactive times in addition to the CPU inactivity resulting from the serialisation. Overall, the wall time increased by 20% for double the workload, making OC a worthwhile optimisation technique for this case. This example showcases that OC is useful as a latency hiding method.

In conclusion, it can be seen that OC increases the memory requirements and CPU usage. This is the trade-off that has to be considered before implementing OC. In the above examples, the overall wall time increased in both OC cases. What has been shown, is that the increase in wall time is in relation to the proportion of CPU inactivity during the not overcommitted scenario. In principle a scenario in which the wall time does not increase, such as a very low CPU activity in the no-OC case, or increases by exactly the added work, such as 100% CPU usage in the no-OC case, can be possible.

These profiles open up several questions that have been investigated in Section 8.3.

- Which workflows and scenarios can OC be useful for?
- Does the RAM configuration of the existing machines allow for OC, or how much RAM is required?
- How many processes should be overcommitted?
- Should the processes be doubled, as in the above example?

A cost/benefit analysis is performed in Subsection 8.4.2.

8.3 Measurements

8.3.1 AthenaMP combinations

With AthenaMP, OC is easily done on an individual VM. It was already found, that running only one AthenaMP instance in OC mode does not yield any benefits in the case of local data. This is because it does not influence the overall profile, see Subsection 8.2.1. Running multiple instances in parallel, for example, two as in Subsection 8.2.2, gives the expected improvement in CPU efficiency. How many instances and what combination gives the best result, remains to be answered. Possible scenarios are plentiful, e.g. running as many instances as possible as parallel processes. On a 4-core VM, adding one additional process could look like: 4+1; 3+2; 3+1+1; 2+2+1; 2+1+1+1 and 1+1+1+1+1 The numbers indicate how many parallel processes are run in one AthenaMP instance. Different instances are separated by a plus sign.

In addition to that, there can be more than five parallel processes on a 4-core VM.

Some scenarios can already be excluded beforehand. If there are too many parallel processes, the RAM requirement becomes too high, whereas the CPU efficiency does not benefit.

For the many AthenaMP instance scenario, the memory footprint is larger than for the scenario with fewer instances. This is due to the fact that the processes share part of their memory to reduce redundancies. AthenaMP increases the sharing, by having the processes share additional data, such as the detector geometry. This additional AthenMP sharing is only taking place for processes within the same instance.

After excluding some scenarios, what remains is to measure and compare. The results of the initial measurements are shown in Figure 8.6.

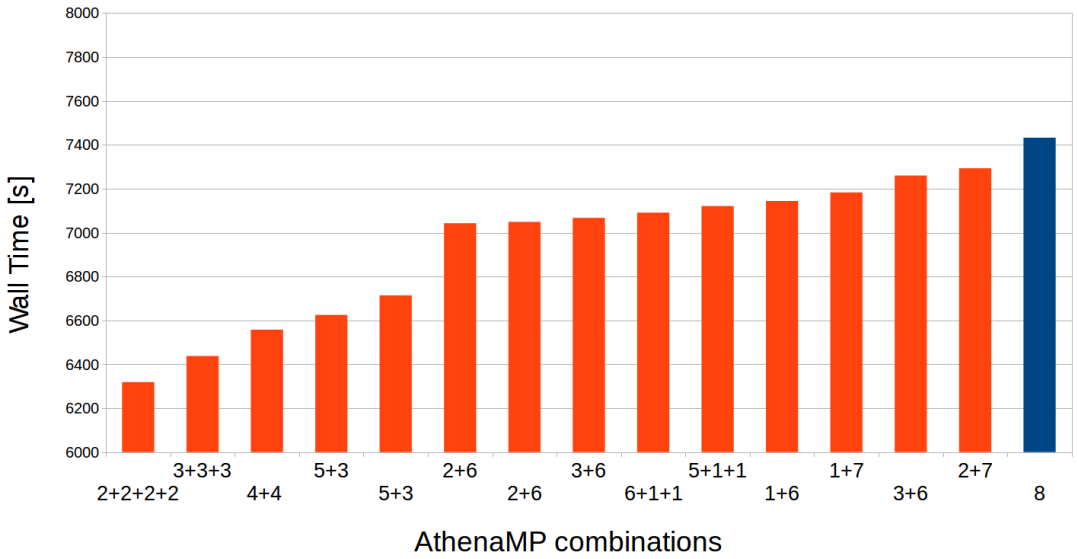

Figure 8.6: Runtimes of different AthenaMP OC configurations. The standard eight processes on the 8-core configuration is marked in blue. Note: for better readability, the wall time does not start at zero.

At first glance, this looks surprising as all possible scenarios seem to take less time than the standard configuration. Even the ones that are not overcommitted, such as 7+1, for example, took less time. The explanation can be found in the way the jobs are executed and in the resulting files. First of all, a fair comparison has to be ensured, the interpretation of the results is done afterwards. Evoking AthenaMP once with eight parallel processes, delivers the results in a single merged file. In the case of the 7+1 invocations, the results will be split among two files, one containing $\frac{7}{8}$ of the results, the other $\frac{1}{8}$. These two files would have to be merged in addition.

If further merging steps are considered for the multiple AthenaMP instance scenarios, the durations change accordingly. This means that the wall time of, for example, the

7+1 scenario has to be increased by the time it takes to merge the two resulting output files. This additional merging time, that has to be added to all scenarios with multiple output files, is extrapolated from the standard scenario[1]. In Figure 8.7, the wall time corrections of the additional merging have been applied.

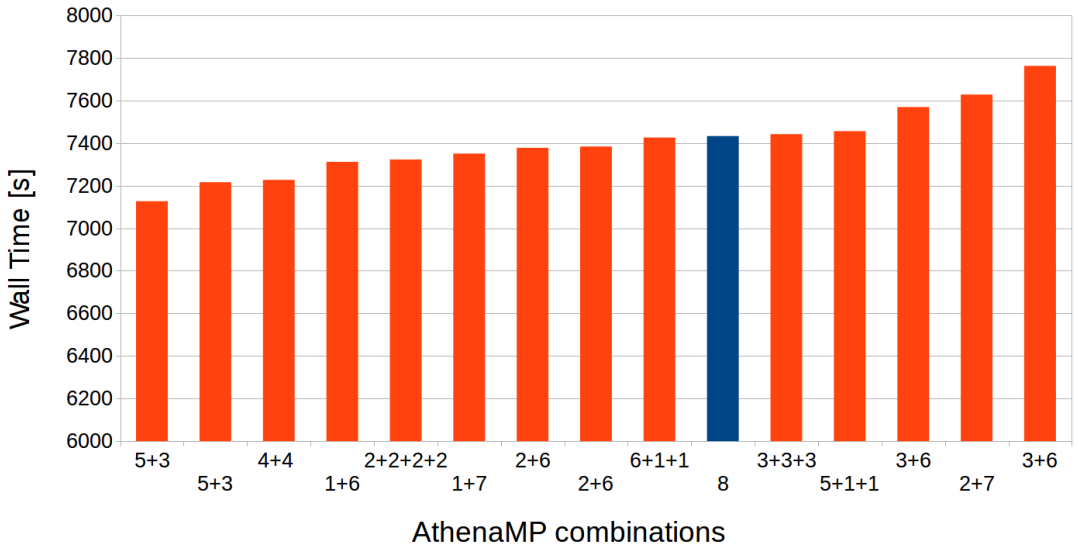

Figure 8.7: Runtimes of different AthenaMP OC configurations after applying merging corrections. The standard eight processes on the 8-core configuration is marked in blue. Note: for better readability, the wall time does not start at zero.

What can be seen, is that the order of the combinations change, as some would have to undergo more merging than others. More importantly, the difference between the standard eight process scenario and the others shrinks significantly. Some scenarios are even taking longer than the standard one.

In order to avoid differences with the merging, a workaround has been found. Instead of comparing single jobs with each other, a set of jobs was compared. This was done in the following way: in the usual scenario, four jobs run successively and produce four output files. Therefore, the output will be the same for the 2+2+2+2 scenario that is processing four times the workload of a usual job. In both cases four output files, containing roughly 3000 physics events each, are produced. These are not merged further. Figure 8.8 compares the runtime, divided by four, of these scenarios.

Without the need to apply merging corrections, the difference in the runtime is sig-

[1]Initially it was attempted to compute and measure the additional merging steps manually. Due to configuration errors, they could however not be executed correctly and the results were inconsistent with the original 8 core AthenMP job. After consulting experts, it was decided that extrapolating would be the better alternative.

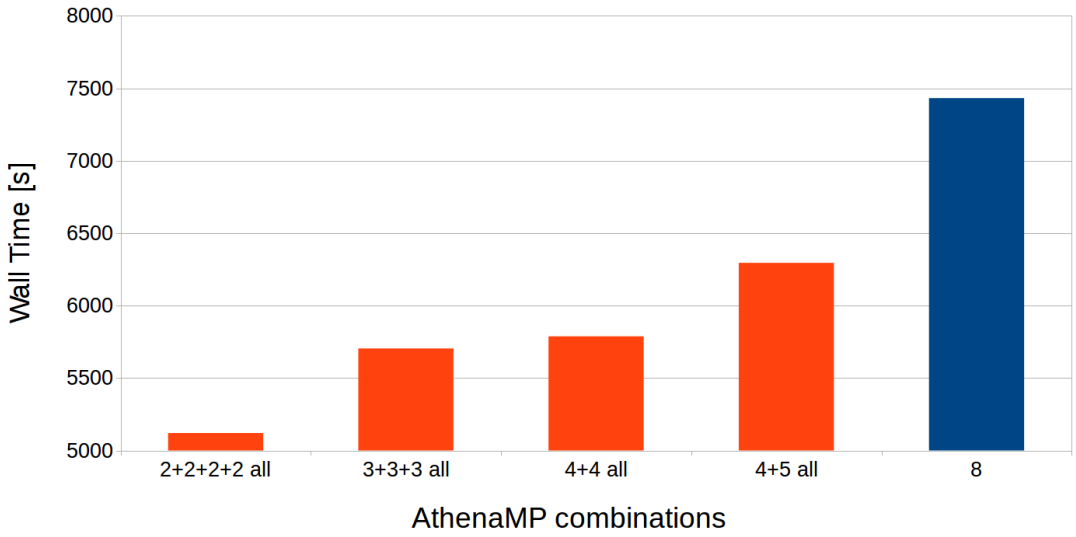

Figure 8.8: Runtimes of different AthenaMP configurations after applying the workaround. Not all scenarios are overcommitted. Note: for better readability, the wall time does not start at zero.

nificant. This goes even for cases, in which technically no OC is performed (2+2+2+2 and 4+4). The speed-up in the non-OC cases can be explained by the additional flexibility and by the fact that multiple cores are active during the merging. This becomes apparent when looking at the profile of the 2+2+2+2 scenario, shown in Figure 8.9.

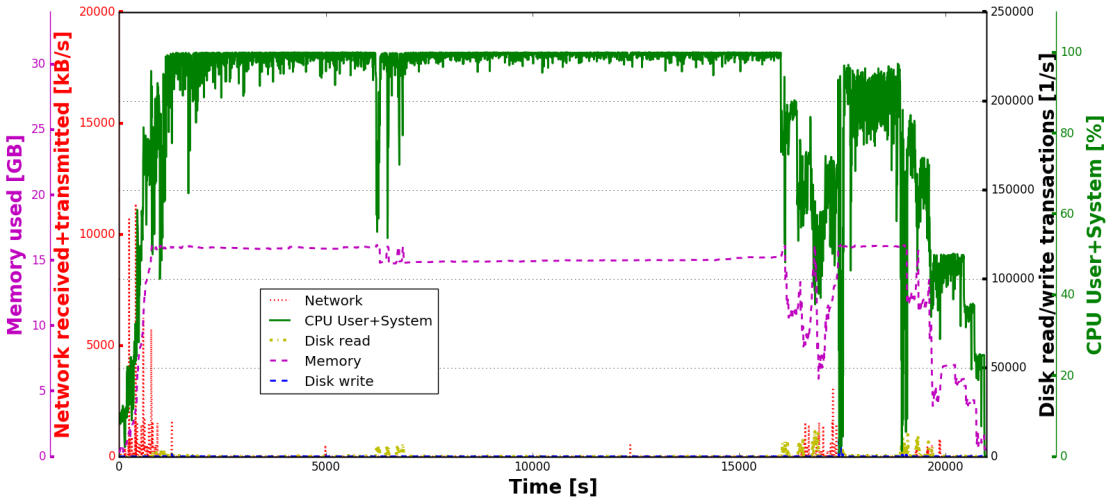

Figure 8.9: Profile for the 2+2+2+2 scenario. Note: no logarithmic scale is used for the network and disk activity.

Especially towards the end, after 18000 s, and in between the RAWtoESD and ESDtoAOD steps, between 16000 s and 17000 s, the CPU usage stays high. In the usual scenario, the CPU usage would drop down to 12.5% during these serial merging steps. This can be seen, for example, after 9000 s in Figure 8.4.

What has to be highlighted at this point, is that the single core merging step does not reach the disk read/write limit[2]. This is why multiple CPU cores can execute different single core merging processes at the same time without being stuck in I/O wait.[3] Finally, this explains the ordering of the processes in Figure 8.9. The higher the number of AthenaMP instances, the better the CPU utilisation during the merging period, and the shorter the overall wall time. This is limited by the memory footprint that increases with each additional instance.

In conclusion, what has been shown in this section is that there are many possibilities to overcommit. Due to the particulars of the merging, the comparison between all scenarios is not trivial. The combinations that provide the highest event throughput, are the ones that exploit the serialisation part the most. Instead of testing all possible combinations, a better way to use the CPU cycles fully during the serialisation is described in Subsection 8.3.3.

8.3.2 Overcommitting for latency hiding

After looking at the profiles of overcommitted VMs, it became apparent that OC can be used to compensate negative effects from higher latencies. There are multiple scenarios in which remote access is beneficial, like in partial reads, or is already used, like in CMS. This has been investigated in more detail. The results can be found in Table 8.1.

Number of processes	**RAM [GB]**	**Input data location**	**Overall node throughput [s/event]**	**Overcommit improvement [%]**	**Improvement to standard config [%]**
8	32	BNL	4,19 ± 0,05		-32
2x8	32	BNL	2,55 ± 0,01	39	**19**
8	16	BNL	4,31 ± 0,08		-36
2x8	16	BNL	3,51 ± 0,08	19	-11
8	32	local	3,07 ± 0,04		**3**
2x8	32	local	2,24 ± 0,01	27	**29**
8	16	local	3,17 ± 0,09		0
2x8	16	local	3,33 ± 0,01	-5	-5

Table 8.1: Overcommittment results summary [117].

Adjacent rows with a similar background colour depict the same infrastructure config-

[2]The VM on which the profiling was done, used a hard disk. The disk limit was not reached.
[3]ATLAS is currently working on removing the limitation of writing from a single core to a single file, see Subsection 8.5.

urations. The difference is that the upper rows within a pair show the situation without OC, while the lower rows are overcommitted by a factor of two, including double the workload. This factor is chosen in order to circumvent the merging issue, that was already discussed in Subsection 8.3.1.

Adjacent pairs of rows with the same colour, differ with regards to the available RAM. The pairs of rows alternate between 16 GB, which is the standard Grid configuration for an 8-core machine and 32 GB. In this case, 32 GB was the maximum that the VM had available. It was chosen in order to make sure the jobs do not run into any memory limitation, meaning to avoid heavy swapping.

In addition, in the top four rows, the input data were located at BNL. The data were read from there event-by-event during the workflow execution. This is compared to the bottom four rows, depicting the standard scenario of local input data. Local in this case means on the VM's disk. The download duration from a local storage, such as EOS or DPM, to the VM's disk is very small, with regard to the job duration, so it is neglected.

The results from Table 8.1 show that the best event throughput is not achieved by the standard Grid configuration. Instead, the OC scenarios with additional memory are 19% faster, with input data at BNL, and 29% faster, with local input data.

It can be also highlighted that OC works as a latency hiding mechanism. An improvement through OC can be observed in both remote data measurements. This effect was almost twice as high with more RAM, 39% compared to 19%.

In the standard RAM configuration, OC was not enough to completely compensate the remote data slow down. Even with OC it was still 11% slower than with local input data.

In the standard Grid configuration, OC has a negative impact on the performance (-5%). When additional memory is available however, a significant performance gain ($+27\%$) can be observed.

In conclusion, OC brings advantages to the remote data reading scenario. Since the benefits for OC depend on the available memory, the ideal RAM-to-core ratio has to be found. One possible approach to find the correct ratio, was already shown in the RAM investigation undertaken in Section 5.3.2. In addition, the question of how many processes to overcommit remains. These would have to be tested against every RAM configuration, as the number of processes influences the RAM requirement.

An alternative is to use the WIM, see Section 8.4. It can correlate the RAM with the number of processes and even include the Cloud cost.

8.3.3 Scheduling

By now it should be clear, that the merging step is the major contributor to the CPU inefficiencies. Instead of using trial and error, like in Subsection 8.3.1, the merging can be targeted specifically for improvement through OC. The result is a scenario that avoids the large CPU inefficiencies of merging, while at the same time making the

processing beforehand more efficient.

Instead of doing the merging consecutively, it can be done in parallel. A second reconstruction job would start its processing as soon as the first job goes into merging, and so on. In principle, this can be achieved through intelligent scheduling.

Alternatively, reconstruction jobs could be run consecutively, without any merging. In addition, merging jobs would be run in parallel to that. The first merging job could only start after the first reconstruction finishes.

In both scenarios, there is an overhead at the beginning of the exercise, when there is no unmerged data on the VM, in which no OC would take place. In the Grid scenario, an additional boundary is the lifetime of the pilot job. When the pilot job expires, this OC processing chain is stopped, and the last jobs are killed. The maximum lifetime of a pilot job is three days.

A reconstruction job duration of 9711 s, a job duration without merging of 5871 s and a job duration with the OC merge in parallel of 6334 s highlights that a good improvement can be achieved. In a 72 hour job slot, 38 jobs with an OC merging in parallel could be executed, whereas in the usual scenario it would only be around 27. This is around 40% more workload that can be performed on the same CPUs. One caveat is that the available memory for this measurement was 32 GB, instead of 16 GB.

8.4 Model predictions

In this section, the WIM is combined with the OC, which can then be combined with the flexible hardware configuration of the Cloud computing.

8.4.1 Overcommitting in the model

In Subsection 8.3.2 it was mentioned, that the WIM can be used to make predictions for the OC. In order for the model to make a meaningful prediction for OC, some tweaks had to be incorporated. The specific OC input values that are introduced below, can simply be set to zero in the usual (non-OC) case.

Overcommit improvement time

The overcommit improvement time ($OC\_improve\_time$) describes the gain achieved by the OC. In simple terms, it can be described as the wall time difference between the non-OC and the OC scenario. This can be visualised easily on a single-core machine. For example, if running two processes consecutively takes 100 s, but running them in parallel would take 85 s, then the $OC\_improve\_time = (100 - 85)\ \mathrm{s} = 15\ \mathrm{s}$.

A more complex scenario was shown in Figure 8.2. There, the $OC\_improve\_time = (40 - 34)\ \mathrm{s} = 6\ \mathrm{s}$. The 34 s represent the wall time. The 40 s represent the wall time it would take, if there was no OC benefit, as can be seen from Figure 8.1.

The OCF gives the percentage of the OC workflows that were computed using the unused cycles of the non-OC scenario. Using the OCF, the $OC\_improve\_time$ can be calculated as follows:

$$OC\_improve\_time = OCF * (nr\_processes - Nr\_Cores) * \\ (CPU\_consumption\_time + IO\_Wait\_Time + Idle\_Time + \\ Overhead\_Time)/Nr\_Cores \quad (8.3)$$

where the $nr\_processes$ is the total number of processes and Nr_cores is the number of available CPU cores. The term in brackets in the second line represents the non-OC wall time.

In the example shown in Figure 8.2, the $OC\_improve\_time = 0.75 * \frac{5-4}{5} * 32 \text{ s} = 6 \text{ s}$, which agrees with the earlier result.

This gain has to be accounted in the workflow duration.

Overall workflow duration

The overall duration of the workflow ($Overall\_WF\_Time$) is now calculated by additionally applying OC, see Equation 8.3, to the standard workflow time, see Equation 6.10:

$$Overall\_WF\_Time = (WF\_Time - UC\_Idle\_Time - OC\_improve\_time)/Nr\_Cores \quad (8.4)$$

where UC_Idle_Time is the undercommittment idle time, see Subsection 4.6.5. In the end, either the OC or the UC time has to be zero. For workflows configured like on the Grid, both are zero.

8.4.2 Result

With these additional implementations, the WIM is able to make predictions for OC. One use case that was already mentioned in Subsection 8.3.2, is to find the best memory/OC combinations. Furthermore, all other WIM functionalities can be used.

A concrete example would be, to combine OC with the flexibility of a Cloud provider. In this example, a flat budget of 1000 CHF is available to procure Cloud resources that will run ATLAS raw data reconstruction. The detailed input parameters can be found in the Appendix, see Subsection A.4.1. Instead of immediately purchasing VMs that have the standard configuration of eight CPU cores with 16 GB of RAM, the WIM is used to determine the best configuration. Taking the Cloud prices of CloudSigma and the reconstruction workflow parameters as Model input, the following graph is created, see Figure 8.10.

According to the specifications, the graph is in 3D, with the number of parallel processes and the available RAM as parameters.

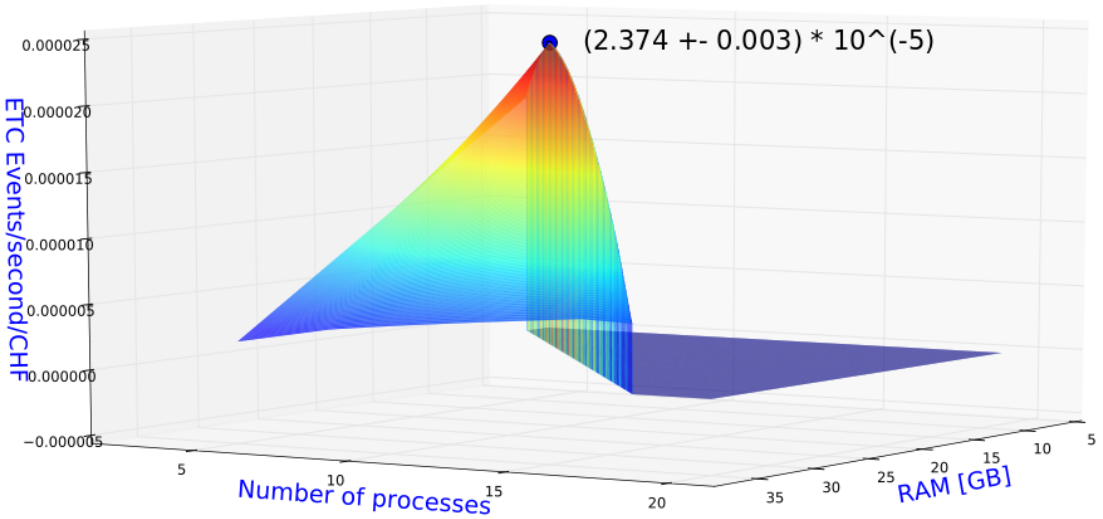

Figure 8.10: Graphical display of the Model output. The maximum is highlighted by the blue dot [117].

The shape can be explained by the boundary conditions. It can be seen that when decreasing the RAM, a sharp edge where the graph drops off is reached. This is the cut-off that happens due to heavy swapping, if there is not enough memory available. It is also logical that unused CPU cores lead to a less-than-optimal yield, which explains why a higher undercommitting has a downward trend. The reason why a rising number of processes can have a negative impact on the event throughput is that the memory requirement rises with the processes. There is a flat budget, so additional RAM lowers the budget for the other components, in this case the CPUs. More RAM translates directly to fewer CPUs.

The position of the maximum value on the x-y-plane provides the best RAM/processes configuration, that maximises the number of produced events while minimising the time and the cost. For 1000 CHF in 10000 s the model predicts that 23740 ± 30 events can be reconstructed. The result of $\sim 2.374 \pm 0.003$ events/second/CHF lies at 14 GB of RAM, indicating that the current 2-to-1 ratio of RAM [GB] to CPU [cores] is not the best configuration. With this improved configuration, a 12% higher event throughput is achieved, compared to the standard scenario. This result is valid for the chosen reconstruction workflow and under an OC of nine processes per VM.

The same graph can be created for several Cloud providers to get a comparison between their offers. The highlighted maximum value would then be compared between the different providers. The comparison would then automatically compare the optimal configurations for each provider. Since the pricing models are quite different between the providers, the optimal configuration can be different between the providers. This way a fairer comparison is achieved than just comparing one exact configuration between

all providers.

8.5 Conclusion

This chapter showed the possibilities of overcommittment. The focus was on OC at the job level, on top of the OC done at the infrastructure level. It has been found, that OC works well as a latency hiding method. Given enough RAM, additional scenarios can benefit from OC, like workflows that heavily use the disk. Therefore, OC is especially attractive for Cloud computing, where the infrastructure is flexible, especially the memory per CPU core, and the infrastructure can be adapted to the workflow.

Furthermore, the computing capacity of serialised periods in the workflows can be exploited. This is achieved by executing a heavier load, or by adapting the workload to the workflow profile, for example, scheduling a merge in parallel to a processing.

What has also been demonstrated is that there is a large potential for improvement, regarding the single core merging step within the ATLAS reconstruction. ATLAS chose a different path than OC to improve the workflows for now. Instead of directly using OC to lose the merging inefficiencies, the workflow was reworked. The merging will be avoided by having the parallel AthenaMP processes write their outputs directly to a single file. This approach is called the "SharedWriter". At the time of writing, this has not gone into production yet. It may turn out, that this new workflow would also benefit from OC, due to its I/O activity.

With some modifications to the WIM it provided a good way to compare overcommitted scenarios with each other, even taking into account different Cloud providers. The Model then indicated the best configuration to run a workflow within a given Cloud, by maximising the ETC, or the event throughput.

CHAPTER 9

Cloud viability evaluation

The WIM, that was introduced in detail in the previous chapters, can be used to evaluate whether Cloud computing is a viable alternative to Grid computing. Grid computing in that context means the purchasing and operating of hardware in the many computing centres, including the personnel. Cloud computing means renting computing resources from a commercial provider.

In Subsection 4.2.2 the Cloud pricing development was described, highlighting the downward trend. It is however difficult to understand how workflows would perform within a Cloud. Looking only at the price of a Cloud site is therefore not sufficient to evaluate whether moving to the Cloud is beneficial. For that reason the WIM was introduced, which combines the pricing information with workflow performance estimations.

One of the big advantages when using the WIM is the possibility to compare many different Cloud configurations with each other, as was shown in Chapter 8. As was also shown before, not all workflows have the same infrastructure requirements, e.g. CPU core to RAM ratio. In contrast to the Grid, where the resources are static, the Cloud can be provisioned flexibly. The model in the end provides a result that includes the best Cloud configuration for the given workflows, thus making sure that the comparison is as fair as possible. A Cloud provider that gives the best price per event ratio for one workflow is not necessarily the best choice for a different workflow.

In this section, not every possible combination of all the Cloud providers and all Grid centres is evaluated. The inputs will change on a case by case basis, which can change the results and conclusions. Some of the reasons highlighting the importance of investigating the viability of Cloud computing individually, are given in the following.

First of all, the Cloud prices change over time, which can make the given results outdated within a short time. More importantly, the Cloud prices depend heavily on the negotiation skill of each procurer, discounts, as well as the type of services included

and the duration of the procurement.

Similar considerations can be made for the Grid sites, which are located around the Globe and for which the total cost of ownership are very different. The different cost factors are amongst others, the cost for electricity, the power and cooling efficiency of the Grid centre, the personnel cost and the negotiated prices for the hardware, which is cheaper if bought in bulk.

Grid prices

In order to understand the differences between the Grid and the Cloud, a Grid site has to be chosen, that acts as a reference. The CERN computing centre is chosen as a representative example, as it is at centre of the WLCG, hosting computing capacity for all major LHC experiments.

The total cost of ownership of a server is taken from the recent paper "Cloud computing, markets and costs" by Bernd Panzer-Steindel, the CERN IT CTO. As of the time of writing, it has not yet been published. The total yearly cost of a 40 core compute server is given in that paper as 1570 €. (Consisting of: Physical server = 900 €; Rack = 10 €; Power distribution unit = 20 €; Network infrastructure = 30 €; Floor space = 70 €; Electricity = 135 €; Personnel cost = 405 €) In order to account for memory differences, the estimation in the paper increases to 0.006 € per core hour (rounded up, no VAT included). For an eight core VM this leads to 0.048 € per VM hour.

Cloud prices

There is a plethora of choices on the side of the Cloud providers, each with their own specifications. Amazon is the market leader and provides the most attractive prices on its spot market (0.014 € per core hour). Unfortunately, no benchmark data could be collected for Amazon. In the following, the results of papers doing a cost comparison between a Grid centre and the Cloud (Amazon AWS) are summarised.

ATLAS is using and considering Cloud resources for several years. In a paper from 2010 [136], which compares the Grid centre cost in Germany with the Amazon cost, the Grid centre is more cost effective. In a paper from 2014 [137], the conclusion is that the Amazon spot prices are competitive compared to the cost of the RHIC and ATLAS computing facility at BNL.

There are results from CMS, which used the Amazon AWS infrastructure to run some of their workflows. The cost of AWS was compared to the cost of the resources at Vanderbilt University in this paper from 2012 [138]. The result shows that if the University resources are utilised above 14.5% per year, they are more cost effective.

In a more recent paper from 2017 [139], the cost of a CPU core hour for CMS at Fermilab is given as $0.009 + -25\%$. The power of a CPU in Amazon is at the level of 97% compared to the ones at Fermilab, according to the benchmark in this paper [67].

The paper [67] concludes, that the hardware at Fermilab is more cost effective than the AWS offer.

It has to be kept in mind, that the spot market is volatile and it is not guaranteed that resources are available for the stated price when needed.

For the WIM, the providers that were investigated in detail, were the three providers described within the HNSciCloud context in Chapter 7. The prices were obtained from their web pages. Event generation uses the single core setup, whereas the other workflows are estimated via the 8 core VMs.

Currently the price for an IBM B1.8x16x100 instance with 8 CPU cores, 16 GB of RAM and 100 GB of storage is 0.31 € per hour[1]. The single core VM setup is called B1.1x2x25 has 1 CPU core, 2 GB of RAM and 25 GB of storage space and costs 0.039 € per hour.

The T-Systems instance called CO II-4 comes with the same configuration, namely 8 CPU cores, 16 GB of RAM and 100 GB of storage. The price is 0.406 € per hour[2]. The single core VM setup is called CO II-1 with 2 GB of RAM and 25 GB of storage and costs 0.023 € per hour.

For Exoscale, a slightly different configuration has to be considered, as it is not as flexible as the other providers. The "huge" instance consists of 8 CPU cores, 32 GB of RAM and 100 GB of storage and costs 0.332 € per hour[3]. The only possible VM configuration that has 2 GB of RAM comes with 2 CPU cores. The price of 0.026 € per hour of the 2 core configuration called "small" is used to calculate the single core workflows. It has 50 GB of storage space.

9.0.1 Model results

In Table 9.1, the WIM comparison between the different Cloud providers and CERN is shown. The result is given as Events/Euro. It gives the maximum number of events that can be processed per Euro. In this case, the 'Time' from ETC does not add any benefit, as the WIM was exercised as a Cloud procurement with a fixed contractual time limit.

What can be seen immediately, is that the Grid solution "CERN" is much more cost effective than the Cloud offers. This shows, that the performance differences between the sites are outweighed by the cost. Possible performance benefits are on a scale that is too small to compensate for the much lower cost of the Grid hardware at CERN.

The errors that are estimated from the model are excluded, as they are too low, in the order of a few percent. A better estimation can be gained from looking at the comparison between the model results and the large scale measurements. In the end, even if the largest deviations are considered for the results, they cannot possibly change

[1] The IBM prices were taken from their web interface. (02.04.18) https://www.softlayer.com/cloud-computing/bluemix/Store/orderComputingInstance

[2] The T-Systems prices were taken from their web interface. (02.04.18) https://cloud.telekom.de/en/infrastructure/open-telekom-cloud/price-calculator/

[3] The Exoscale prices were taken from their web interface. (02.04.18) https://www.exoscale.com/pricing/#/compute/huge

Model result [**Events/Euro**]	**CERN** Grid	**Exoscale** Cloud	**IBM** Cloud	**T-Systems** Cloud
EvGen	132881	50785	23358	37560
MC Sim	2315	782	383	255
Reco 1	18458	4058	2769	
Reco 2	16904	4810	2942	1521
Reco 3	38958	10306	5407	4130
Reco 4	34716	9431	5540	3872
Reco 5	54708	13947	8189	5763
Reco 6	39413	10262	6113	4041
Reco 7	36591	7844	5390	2845
Digi Reco 1	6775	1915	914	746
Digi Reco 2	10246	2559	1154	646

Table 9.1: Model results according to the price schema from the webpage and the VM performance from the benchmarks performed during the HNSciCloud prototyping phase.

the conclusion. This is due to the fact that the pricing differences outweigh performance variations, even if they are at the order of 30%.

9.1 Conclusion

According to the prices that the Cloud providers publish, they are not yet competitive, compared to purchasing and operating computing hardware within the WLCG.

The WIM results are in line with the findings of the previously summarised Cloud viability studies. The difference is that the WIM findings show a larger gap between the Grid and the Cloud resource cost. This is a result of the lower total cost of ownership of the hardware at CERN compared to e.g. Fermilab and the higher cost of the investigated Cloud offers compared to the Amazon spot market.

The fact, that the difference in cost between the Cloud and Grid resources varies, highlights the fact that these studies have to be performed individually. This is because the costs vary for every site. The WIM is easily adaptable to include the use cases of different Grid sites and Cloud offers.

What can also be seen, is that the Cloud prices became more competitive over the last 8 years. This development has however stalled.

CHAPTER 10

Summary and Outlook

During the course of this thesis, a model (WIM) was created, which predicts the performance of a given set of workflows on an arbitrary infrastructure. The goal was to understand the behaviour of data intensive workflows within the Cloud in order to assess its feasibility. The WIM includes the possibility to correlate the performance prediction with cost considerations, so a comparison between different Cloud offers and the Grid is possible.

In order to create the WIM, the general workflow behaviour had to be understood. This is important when choosing a minimal set of model input parameters which needs to be able to describe the workflows with a high accuracy. Furthermore, this lead to the discovery of optimisations to the workflows as well as the infrastructure configuration.

In the early stages, the model was already used to correctly predict that the 10 Gb/s bandwidth link between a 4000 CPU core strong Cloud site and CERN would be sufficient to support running ATLAS reconstruction workflows efficiently. This shows the versatility of the WIM, in which even typical input parameters can become the result.

At a later stage, the WIM was improved to be able to vary and correlate different input parameters. This gave the possibility to find optimisations. The modelling and comparison of different infrastructure workflow combinations can help to avoid many time consuming measurements.

One of the optimisation techniques that was found and investigated alongside the WIM development is the overcommittment (OC). OC means to run additional workload on a machine, in order to fill unused CPU cycles. Through the profiling of the workflows, it became clear that raw data reconstruction could benefit from OC during its serial merging phase. The trade-off is an increased memory footprint. Intensive tests and modelling were performed in order to quantify the usefulness of OC within the Grid and the Cloud. The best result was achieved when using OC in conjunction with re-

construction workflows that retrieve the input data from a storage that was located far away. The improvements in this scenario ranged up to an increased event throughput of 39%.

From there on, the model was used to make predictions within the Cloud. The flexibility of the infrastructure, when using Cloud computing can be better exploited through the usage of the model. One benefit is that the WIM provides the optimal site configuration along with the maximal achievable event throughput. This makes a better comparison between different Cloud offers possible, as only the best event per cost values are compared. This configuration should then also be procured and used.

The WIM was intensively validated on several different infrastructures and running a broad spectrum of ATLAS workflows. It was demonstrated that the predictions are accurate within a 20% margin, but also sensitive to heterogeneous infrastructures and to differences between the benchmark and the workflows.

In the big picture, the WIM was used to evaluate the viability of Cloud computing, compared to the classical approach of purchasing and operating hardware. At this point in time, Cloud computing is less cost effective than operating a classical WLCG Grid centre.

Looking towards the future, it is good to see, that ATLAS already picked up on the inefficiencies that were found during this thesis. An improvement to all AthenaMP workflows is under development, which is called the "shared writer". It is supposed to avoid the serial merging steps, that happen during the raw data reconstruction. The shared writer is only one possibility to optimise the workflows and it remains to be seen if OC can improve on even these workflows. It is possible, that the shared writer leads to an increase in the CPU idle and I/O wait time footprint, which can be optimised by OC.

Furthermore, OC improves workflows with remote data input, which will become more relevant with increasing data sizes.

While investigating how to best set up a Cloud site, it became clear that the data storage is a personpower intensive task. The native Cloud storage is object storage, which is not directly compatible with the storage solutions of the WLCG. Even within the WLCG, the storage is very maintenance intensive. One vision of the future is that the storage that is distributed over many sites, will be consolidated into fewer dedicated storage sites. The remaining sites would then only operate data caches.

According to the recent price development of the Cloud market, it does not look as if Cloud computing will become more cost effective than operating Grid centres over the next few years. How the situation will have changed for the HL-LHC is not realistically foreseeable, due to the inconsistent development of the Cloud prices. Considering the stagnating prices, Cloud computing will most likely not be the solution that can bridge the estimated HL-LHC computing resource discrepancy. Cloud resources may, however, be used as burst capacity to bring relieve during short termed peaks in demand.

Acknowledgements

I would like to warmly thank Arnulf Quadt, who through countless efforts enabled this work. Without his dedication and support, this work and my participation in the Gentner program, would not have been possible. Arnulf left me the freedom to pursue three years of study at CERN, while at the same time accepting the additional overhead of supervising me over a 800 km distance. I also very much appreciate the opportunities, to present my work at international conferences.

Special thanks also go to Oliver Keeble, who in addition to his responsibilities at CERN, took up the task of supervising me. He helped shaping this thesis through many fruitful discussions, giving me countless insights.

Thank you Gen Kawamura, for your patient supervision. Your contributions were always helpful and are greatly appreciated.

In addition I would like to thank Clara Nellist for taking the time to proof read this thesis.

Thanks also goes to Jörn Große-Knetter, who agreed to be part of my thesis committee.

Furthermore, I would like to thank Ramin Yahyapour, who agreed to be a referee of this thesis.

I would like to thank all of the people at the II. Physics Institute, for their support.

Last but not least, thanks goes to my friends and family, on whose support I can always count.

This work has been sponsored by the Wolfgang Gentner Programme of the German Federal Ministry of Education and Research (grant no. 05E12CHA).

Bibliography

[1] C. Patrignani, et al., *Review of Particle Physics*, Chin.Phys. **C40**, 100001 (2016)

[2] S. L. Glashow, *Partial Symmetries of Weak Interactions*, Nucl. Phys. **22**, 579 (1961)

[3] A. Salam, *Weak and Electromagnetic Interactions*, Conf.Proc. **C680519**, 367 (1968)

[4] S. Weinberg, *A Model of Leptons*, Phys. Rev. Lett. **19**, 1264 (1967)

[5] The CDF Collaboration, *Evidence for $B_s^0 \to \phi\phi$ Decay and Measurements of Branching Ratio and A_{CP} for $B^+ \to \phi K^+$*, Phys. Rev. Lett. **95(3)**, 031801 (2005)

[6] The ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST **3(08)**, S08003 (2008)

[7] The CMS Collaboration, *The CMS experiment at the CERN LHC*, JINST **3(08)**, S08004 (2008)

[8] The ATLAS Collaboration, *Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC*, Phys. Lett. B **716(1)**, 1 (2012), arXiv: 1207.7214

[9] The CMS Collaboration, *Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC*, Phys. Lett. B **716(1)**, 30 (2012)

[10] The ATLAS and CMS Collaborations, *Combined Measurement of the Higgs Boson Mass in pp Collisions at $\sqrt{s} = 7$ and 8 TeV with the ATLAS and CMS Experiments*, Phys. Rev. Lett. **114**, 191803 (2015)

[11] H. A. Lorentz, *Electromagnetic Phenomena in a System Moving with any Velocity Smaller than that of Light*, in *Collected Papers*, pages 172–197, Springer, Dordrecht (1937)

[12] J. Tuckmantel, *Synchrotron radiation damping in LHC and longitudinal bunch shape*, CERN-LHC-Project-Report-819 (2005)

[13] L. Evans, P. Bryant, *LHC Machine*, JINST **3(08)**, S08001 (2008)

[14] G. Barr, et al., *Particle Physics in the LHC Era*, Oxford Master Series in Physics, Oxford University Press (2016)

[15] G. Apollinari, et al., *High Luminosity Large Hadron Collider HL-LHC*, arXiv:1705.08830 [physics] (2017)

[16] ATLAS Inner Detector Community, *ATLAS inner detector : Technical Design Report, 1*, CERN-LHCC-97-016 (1997)

[17] H. Messel, D. F. Crawford, *Electron–photon shower distribution function tables for lead, copper, and air absorbers*, Kent : Elsevier Science, 2013. - 1519 p. (1970)

[18] ATLAS LARG Unit, *ATLAS liquid-argon calorimeter : Technical Design Report*, CERN-LHCC-96-041 (1996)

[19] ATLAS Tile Calorimeter Collaboration, *ATLAS tile calorimeter : Technical Design Report*, CERN-LHCC-96-042 (1996)

[20] ATLAS Muon Collaboration, *ATLAS muon spectrometer : Technical Design Report*, CERN-LHCC-97-022 (1997)

[21] W. Buttinger, *The ATLAS Level-1 Trigger System*, JPCS **396(1)**, 012010 (2012)

[22] A. Martinez, The ATLAS Collaboration, *The Run-2 ATLAS Trigger System*, JPCS **762(1)**, 012003 (2016)

[23] G. E. Moore, *Cramming More Components Onto Integrated Circuits*, Proc. IEEE **86(1)**, 82 (1998)

[24] C. Walter, *Kryder's Law*, Sci. Am. **293(2)**, 32 (2005)

[25] P. H. Enslow, *What is a "Distributed" Data Processing System?*, Computer **11(1)**, 13 (1978)

[26] I. Bird, *Computing for the Large Hadron Collider*, ANNU REV NUCL PART S **61(1)**, 99 (2011)

[27] A. Kaufmann, K. Dolan, *ESG Lab White Paper: Price Comparison: Google Cloud Platfom vs. Amazon Web Services* (2015)

[28] D. Durkee, *Why Cloud Computing Will Never Be Free*, ACMQueue **8(4)**, 20:20 (2010)

[29] A. Sunyaev, S. Schneider, *Cloud Services Certification*, Commun. ACM **56(2)**, 33 (2013)

[30] C. Kilcioglu, J. M. Rao, *Competition on Price and Quality in Cloud Computing*, in *Proc. 25th Int. Conf. WWW*, WWW '16, pages 1123–1132, Int. WWW Conf. Steering Comm. (2016)

[31] L. M. Vaquero, et al., *A Break in the Clouds: Towards a Cloud Definition*, SIGCOMM Comput. Commun. Rev. **39(1)**, 50 (2008)

[32] P. M. Mell, T. Grance, *The NIST Definition of Cloud Computing*, SP 800-145 (2011)

[33] S. K. Nair, et al., *Towards Secure Cloud Bursting, Brokerage and Aggregation*, in *2010 8th IEEE Europ. Conf. on Web Services*, pages 189–196 (2010)

[34] B. Rimal, E. Choi, I. Lumb, *A Taxonomy and Survey of Cloud Computing Systems*, in *5th Int. Joint Conf. on INC, IMS and IDC, 2009. NCM '09*, pages 44–51 (2009)

[35] T. Guo, et al., *Cost-Aware Cloud Bursting for Enterprise Applications*, ACM Trans. Internet Technol. **13(3)**, 10:1 (2014)

[36] R. Ghosh, V. Naik, *Biting Off Safely More Than You Can Chew: Predictive Analytics for Resource Over-Commit in IaaS Cloud*, in *2012 IEEE 5th Int. Conf. on Cloud Computing (CLOUD)*, pages 25–32 (2012)

[37] I. M. Llorente, *The Limits to Cloud Price Reduction*, IEEE Cloud Comp. **4(3)**, 8 (2017)

[38] J. Scholl, A. Lindgren, *Considering Pigeons for Carrying Delay Tolerant Networking based Internet traffic in Developing Countries*, EJISDC **54(0)** (2012)

[39] M. Armbrust, et al., *A View of Cloud Computing*, Commun. ACM **53(4)**, 50 (2010)

[40] B. Satzger, et al., *Winds of Change: From Vendor Lock-In to the Meta Cloud*, IEEE Internet Comp. **17(1)**, 69 (2013)

[41] M. Factor, et al., *Object storage: the future building block for storage systems*, in *2005 IEEE Int. Symp. on Mass Storage Systems and Technology*, pages 119–123 (2005)

[42] Q. Zheng, et al., *COSBench: A Benchmark Tool for Cloud Object Storage Services*, in *2012 IEEE 5th Int. Conf. on Cloud Comp.*, pages 998–999 (2012)

[43] A. Azagury, et al., *Towards an object store*, in *20th IEEE/11th NASA Goddard Conf. Mass Storage Systems and Technologies, 2003. (MSST 2003). Proc.*, pages 165–176 (2003)

[44] K. Popovic, Z. Hocenski, *Cloud computing security issues and challenges*, in *The 33rd Int. Convention MIPRO*, pages 344–349 (2010)

[45] Z. Xiao, Y. Xiao, *Security and Privacy in Cloud Computing*, IEEE Comm. Surveys Tutorials **15(2)**, 843 (2013)

[46] C. Wang, K. Ren, J. Wang, *Secure and practical outsourcing of linear programming in cloud computing*, in *2011 Proc. IEEE INFOCOM*, pages 820–828 (2011)

[47] T. Ristenpart, et al., *Hey, You, Get off of My Cloud: Exploring Information Leakage in 3rd-party Compute Clouds*, in *Proc. 16th ACM Conf. Computer and Communications Security*, CCS '09, pages 199–212, ACM (2009)

[48] P. Kocher, et al., *Spectre Attacks: Exploiting Speculative Execution*, arXiv:1801.01203 [cs] (2018)

[49] M. Lipp, et al., *Meltdown*, arXiv:1801.01207 [cs] (2018)

[50] M. Armbrust, et al., *Above the clouds: A Berkeley view of cloud computing*, UCB/EECS-2009-28 (2009)

[51] Y. Chen, V. Paxson, R. Katz, *What is new in cloud security*, UCB/EECS-2010-5 (2010)

[52] M. Chetty, R. Buyya, *Weaving computational grids: how analogous are they with electrical grids?*, Comput. Sci. Eng. **4(4)**, 61 (2002)

[53] M. Baker, R. Buyya, D. Laforenza, *Grids and Grid technologies for wide-area distributed computing*, Software: Practice and Experience **32(15)**, 1437 (2002)

[54] I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, IJHPCA **15(3)**, 200 (2001)

[55] I. Foster, C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publisher (1999)

[56] M. Livny, et al., *Mechanisms for High Throughput Computing*, Speedup **11-1** (1997)

[57] J. Shiers, *The Worldwide LHC Computing Grid (worldwide LCG)*, Comput. Phys. Commun. **177(12)**, 219 (2007)

[58] D. Espriu, *Report from the Computing Scrutiny Group*, CERN-RRB-2010-052 (2010)

[59] M. Aderholz, et al., *Models of networked analysis at regional centres for lhc experiments (monarc) phase 2 report*, CERN-LCB-2000-001 (2000)

[60] D. Britton, S. L. Lloyd, *How to deal with petabytes of data: the LHC Grid project*, Rep. Prog. Phys. **77(6)**, 065902 (2014)

[61] E. Martelli, S. Stancu, *LHCOPN and LHCONE: Status and Future Evolution*, JPCS **664(5)**, 052025 (2015)

[62] X. Espinal, et al., *CERN data services for LHC computing*, JPCS **898(6)**, 062028 (2017)

[63] P. Andrade, et al., *Review of CERN Data Centre Infrastructure*, JPCS **396(4)**, 042002 (2012)

[64] D. Cameron, The ATLAS Collaboration, *Exploiting opportunistic resources for ATLAS with ARC CE and the Event Service*, JPCS **898(5)**, 052010 (2017)

[65] C. Adam-Bourdarios, The ATLAS Collaboration, *ATLAS@Home: Harnessing Volunteer Computing for HEP*, JPCS **664(2)**, 022009 (2015)

[66] C. Adam-Bourdarios, The ATLAS Collaboration, *Volunteer Computing Experience with ATLAS@Home*, JPCS **898(5)**, 052009 (2017)

[67] L. Bauerdick, The CMS Collaboration, *Experience in using commercial clouds in CMS*, JPCS **898(5)**, 052019 (2017)

[68] R. P. Taylor, The ATLAS Collaboration, *Consolidation of cloud computing in ATLAS*, JPCS **898(5)**, 052008 (2017)

[69] G. Cancio, et al., *Experiences and challenges running CERN's high capacity tape archive*, JPCS **664(4)**, 042006 (2015)

[70] A. J. Peters, E. A. Sindrilaru, G. Adde, *EOS as the present and future solution for data storage at CERN*, JPCS **664(4)**, 042042 (2015)

[71] A. J. Peters, L. Janyst, *Exabyte Scale Storage at CERN*, JPCS **331(5)**, 052015 (2011)

[72] A. Alvarez, et al., *DPM: Future Proof Storage*, JPCS **396(3)**, 032015 (2012)

[73] P. Fuhrmann, *dCache: the commodity cache*, in *In 12th NASA Goddard and 21st IEEE Conf. on Mass Storage Sys. and Tech.* (2004)

[74] A. Dorigo, et al., *XROOTD/TXNetFile: A Highly Scalable Architecture for Data Access in the ROOT Environment*, in *Proc. 4th WSEAS Int. Conf. Telecom. and Inf.*, E-INFO'05, pages 46:1–46:6, World Scientific and Engineering Academy and Society (WSEAS) (2005)

[75] P. Calafiura, others., *The Athena control framework in production, new developments and lessons learned*, in *Comp. HEP and nuc. phys. Proc., Conf., CHEP'04, Interlaken, Switzerland, September 27-October 1, 2004*, pages 456–458 (2005)

[76] G. Duckeck, et al., *ATLAS computing: Technical design report*, CERN-LHCC-2005-022 (2005)

[77] T. Maeno, *PanDA: distributed production and distributed analysis system for ATLAS*, JPCS **119(6)**, 062036 (2008)

[78] T. Maeno, The ATLAS Collaboration, *Overview of ATLAS PanDA Workload Management*, JPCS **331(7)**, 072024 (2011)

[79] P. Nilsson, *Experience from a pilot based system for ATLAS*, JPCS **119(6)**, 062038 (2008)

[80] I. Sfiligoi, et al., *The Pilot Way to Grid Resources Using glideinWMS*, in *2009 WRI World Cong. on Comp. Sci. and Inf. Eng.*, volume 2, pages 428–432 (2009)

[81] R. Brun, F. Carminati, G. Galli-Carminati, *From the Web to the Grid and Beyond: Computing Paradigms Driven by High-Energy Physics*, Springer Science & Business Media (2012), google-Books-ID: pRs3w_DoDroC

[82] V. Garonne, et al., *The ATLAS Distributed Data Management project: Past and Future*, JPCS **396(3)**, 032045 (2012)

[83] V. Garonne, The ATLAS Collaboration, *Rucio The next generation of large scale distributed system for ATLAS Data Management*, JPCS **513(4)**, 042021 (2014)

[84] K. De, The ATLAS Collaboration, *Task Management in the New ATLAS Production System*, JPCS **513(3)**, 032078 (2014)

[85] J. Blomer, et al., *Distributing LHC application software and conditions databases using the CernVM file system*, JPCS **331(4)**, 042003 (2011)

[86] J. Blomer, et al., *Status and future perspectives of CernVM-FS*, JPCS **396(5)**, 052013 (2012)

[87] T. Doherty, et al., *TAG Based Skimming In ATLAS*, JPCS **396(5)**, 052028 (2012)

[88] J. Fulachier, The ATLAS Collaboration, *ATLAS Metadata Interface (AMI), a generic metadata framework*, JPCS **898(6)**, 062001 (2017)

[89] S. Albrand, et al., *The ATLAS metadata interface*, JPCS **119(7)**, 072003 (2008)

[90] A. Lenk, et al., *What Are You Paying For? Performance Benchmarking for Infrastructure-as-a-Service Offerings*, in *2011 IEEE Int. Conf. on Cloud Comp. (CLOUD)*, pages 484–491 (2011)

[91] C. Nieke, W. T. Balke, *Monitoring Performance in Large Scale Computing Clouds with Passive Benchmarking*, in *2017 IEEE 10th Int. Conf. on Cloud Comp. (CLOUD)*, pages 188–195 (2017)

[92] M. Michelotto, et al., *A comparison of HEP code with SPEC 1 benchmarks on multi-core worker nodes*, JPCS **219(5)**, 052009 (2010)

[93] S. Blagodurov, A. Fedorova, *In Search for Contention-descriptive Metrics in HPC Cluster Environment*, in *Proc. 2Nd ACM/SPEC Int. Conf. Perf. Eng.*, ICPE '11, pages 457–462, ACM (2011)

[94] D. Duellmann, C. Nieke, *1st results from a combined analysis of CERN computing infrastructure metrics*, JPCS **898(7)**, 072035 (2017)

[95] E. Karavakis, The ATLAS Collaboration, *Common Accounting System for Monitoring the ATLAS Distributed Computing Resources*, JPCS **513(6)**, 062024 (2014)

[96] I. Bird, et al., *Update of the Computing Models of the WLCG and the LHC Experiments*, CERN-LHCC-2014-014 (2014)

[97] G. Behrmann, et al., *A distributed storage system with dCache*, JPCS **119(6)**, 062014 (2008)

[98] A. Agarwal, et al., *dCache with tape storage for High Energy Physics applications*, JPCS **219(7)**, 072024 (2010)

[99] M. Gorman, *Understanding the Linux Virtual Memory Manager*, Prentice Hall (2004), google-Books-ID: ce1QAAAAMAAJ

[100] D. P. Bovet, M. Cesati, *Understanding the Linux Kernel: From I/O Ports to Process Management*, "O'Reilly Media, Inc." (2005), google-Books-ID: h0lltXyJ8aIC

[101] P. Werstein, X. Jia, Z. Huang, *A Remote Memory Swapping System for Cluster Computers*, in *8th Int. Conf. on Parallel and Dist. Comp., App. and Tech. (PDCAT 2007)*, pages 75–81 (2007)

[102] C. Nieke, et al., *Analysis of CERN computing infrastructure and monitoring data*, JPCS **664(5)**, 052029 (2015)

[103] H. Ishii, *Fujitsus Activities for Improving Linux as Primary OS for PRIMEQUEST*, Fujitsu Sci. Tech. J. **47(2)**, 239 (2011)

[104] K. Salimifard, M. Wright, *Petri net-based modelling of workflow systems: An overview*, EJOR **134(3)**, 664 (2001)

[105] J. Yu, R. Buyya, *A Taxonomy of Workflow Management Systems for Grid Computing*, J. Grid Comput. **3(3-4)**, 171 (2006)

[106] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. Berman, P. Maechling, *Scientific workflow applications on Amazon EC2*, in *2009 5th IEEE Int. Conf. on E-Science Workshops*, pages 59–66 (2009)

[107] M. R. Ebling, L. B. Mummert, D. C. Steere, *Overcoming the network bottleneck in mobile computing*, in *Workshop on Mobile Comp. Sys. App.*, pages 34–36 (1994)

[108] M. Zimmermann, The ALICE Collaboration, *The ALICE analysis train system*, JPCS **608(1)**, 012019 (2015)

[109] M. Böhler, et al., *Evolution of ATLAS conditions data and its management for LHC Run-2*, JPCS **664(4)**, 042005 (2015)

[110] The ATLAS Collaboration, *The ATLAS Simulation Infrastructure*, EPJ C **70(3)**, 823 (2010)

[111] S. Agostinelli, et al., *Geant4a simulation toolkit*, Nucl. Instrum. Methods Phys. Res. A **506(3)**, 250 (2003)

[112] J. Kook, et al., *Optimization of out of Memory Killer for Embedded Linux Environments*, in *Proc. 2011 ACM Symp. Applied Computing*, SAC '11, pages 633–634, ACM (2011)

[113] G. Galster, J. Stelzer, W. Wiedenmann, *A new Scheme for ATLAS Trigger Simulation using Legacy Code*, JPCS **513(2)**, 022039 (2014)

[114] D. Adams, et al., *The ATLAS Computing Model*, CERN-LHCC-2004-037/G-085 (2004)

[115] J. Catmore, et al., *A new petabyte-scale data derivation framework for ATLAS*, JPCS **664(7)**, 072007 (2015)

[116] M. Mambelli, et al., *Job optimization in ATLAS TAG-based distributed analysis*, JPCS **219(7)**, 072042 (2010)

[117] G. F. Rzehorz, The ATLAS Collaboration, *Data intensive ATLAS workflows in the Cloud*, JPCS **898(6)**, 062008 (2017)

[118] T. Binz, et al., *TOSCA: Portable Automated Deployment and Management of Cloud Applications*, in *Advanced Web Services*, pages 527–549, Springer New York (2014)

[119] J. Mirkovic, et al., *DADL: Distributed Application Description Language*, ISI-TR-664 (2010)

[120] C. Stewart, K. Shen, *Performance Modeling and System Management for Multi-component Online Services*, in *Proc. 2nd Conf. Symp. Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 71–84, USENIX Association (2005)

[121] A. Snavely, N. Wolter, L. Carrington, *Modeling application performance by convolving machine signatures with application profiles*, in *Workload Characterization, Annu. IEEE Int. Workshop*, pages 149–156, IEEE Comp. Soc. (2001)

[122] I. Chihaia, T. Gross, *Effectiveness of simple memory models for performance prediction*, in *IEEE Int. Symp. on - ISPASS Performance Analysis of Systems and Software, 2004*, pages 98–105 (2004)

[123] S. Kundu, et al., *Application performance modeling in a virtualized environment*, in *HPCA - 16 2010 16th Int. Symp. High-Performance Computer Architecture*, pages 1–10 (2010)

[124] J. O'Reilly, *AweSim: Introduction to AweSim*, in *Proc. 34th Conf. Winter Simulation: Exploring New Frontiers*, WSC '02, pages 221–224, Winter Simulation Conf. (2002)

[125] A. T. Thor, et al., *ViGs: A grid simulation and monitoring tool for ATLAS workflows*, in *2008 Workshop on Many-Task Computing on Grids and Supercomputers*, pages 1–9 (2008)

[126] D. Breitgand, et al., *SLA-aware Resource Over-commit in an IaaS Cloud*, in *Proc. 8th Int. Conf. Network and Service Management*, CNSM '12, pages 73–81, Int. Federation for Information Processing (2013)

[127] M. Dabbagh, et al., *Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment*, IEEE Network **29(2)**, 56 (2015)

[128] C. Reiss, et al., *Towards Understanding Heterogeneous Clouds at Scale: Google Trace Analysis*, ISTC-CC-TR-12-101 (2012)

[129] H. Liu, *A Measurement Study of Server Utilization in Public Clouds*, in *2011 IEEE 9th Int. Conf. on Dependable, Autonomic and Secure Computing*, pages 435–442 (2011)

[130] H. L. Lee, V. Padmanabhan, S. Whang, *The bullwhip effect in supply chains* (1997)

[131] M. Lindner, et al., *Towards automated business-driven indication and mitigation of VM sprawl in Cloud supply chains*, in *12th IFIP/IEEE Int. Symp. on Integrated Network Management (IM 2011) and Workshops*, pages 1062–1065 (2011)

[132] T. Leng, et al., *An Empirical Study of Hyper-Threading in High Performance Computing Clusters*, Linux HPC Revolution (2002)

[133] L. Gilbert, et al., *Performance Implications of Virtualization and Hyper-Threading on High Energy Physics Applications in a Grid Environment*, in *19th IEEE Int. Parallel and Dist. Proc. Symp.*, pages 32a–32a (2005)

[134] J. Elmsheuser, et al., *Improving ATLAS grid site reliability with functional tests using HammerCloud*, JPCS **396(3)**, 032066 (2012)

[135] D. C. van der Ster, et al., *HammerCloud: A Stress Testing System for Distributed Analysis*, JPCS **331(7)**, 072036 (2011)

[136] J.-P. Gehrcke, S. Kluth, S. Stonjek, *ATLAS@AWS*, JPCS **219(5)**, 052020 (2010)

[137] M. Ernst, et al., *Operating Dedicated Data Centers Is It Cost-Effective?*, JPCS **513(6)**, 062053 (2014)

[138] A. Melo, P. Sheldon, *Integrating Amazon EC2 with the CMS production framework*, JPCS **368(1)**, 012007 (2012)

[139] S. Fuess, et al., *The HEPCloud Facility: elastic computing for High Energy Physics The NOvA Use Case*, JPCS **898(5)**, 052014 (2017)

Appendices

A.1 Job specifications

A.1.1 Workflows: Event generation

Fluctuations

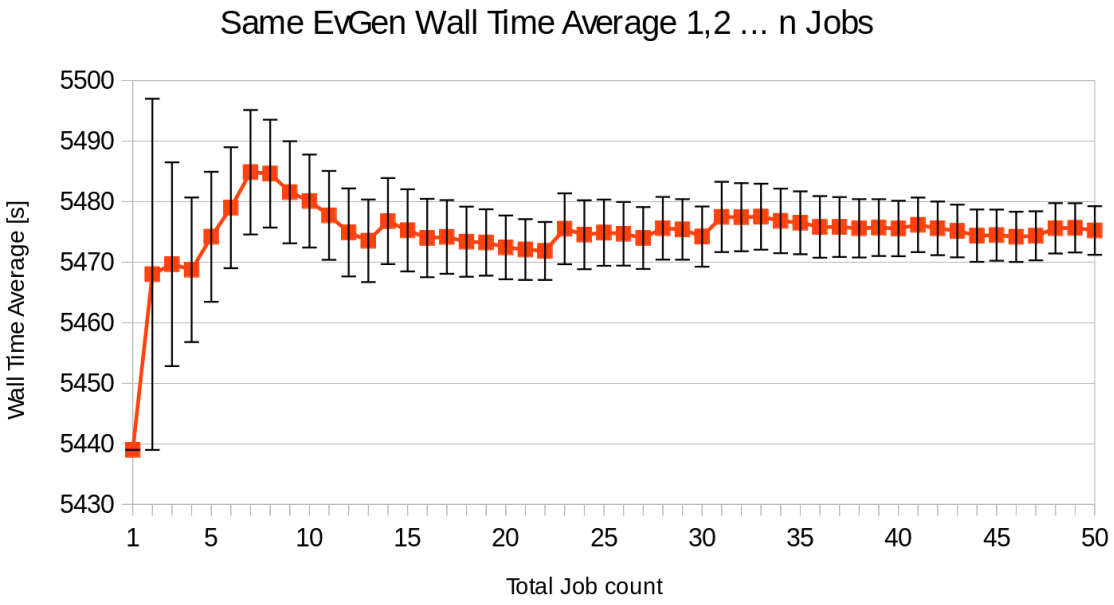

Figure A1: The wall time average over an incrementing number of same event generation jobs (starting at one), at Göttingen. The black error bars show the standard error. Note: in order to improve readability, the y-axis does not start at zero. The input data has been rearranged.

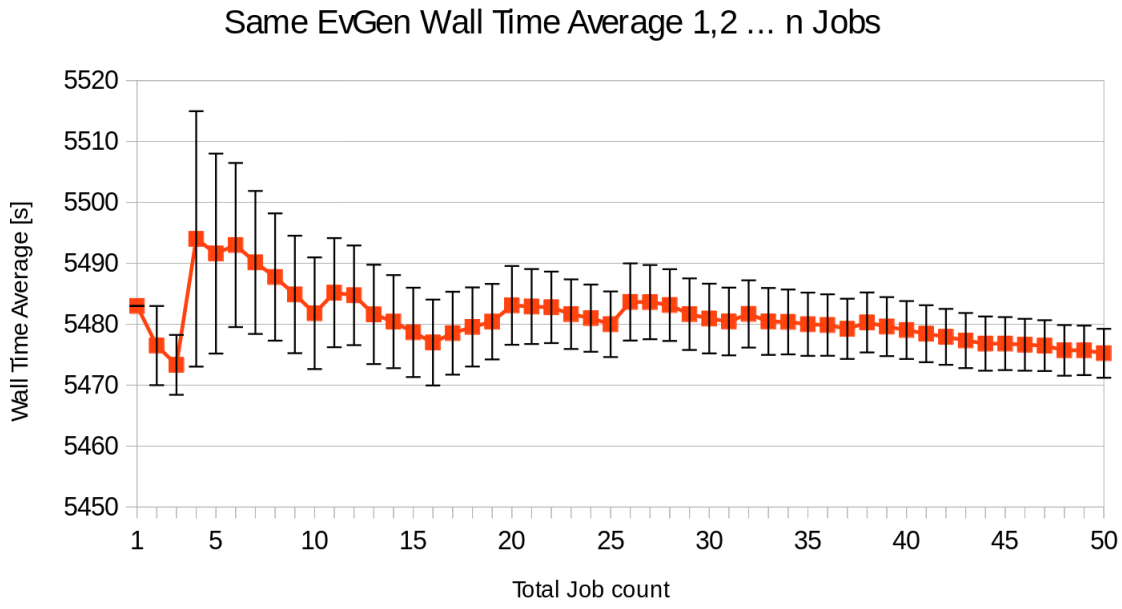

Figure A2: The wall time average over an incrementing number of same event generation jobs (starting at one), at Göttingen. The black error bars show the standard error. Note: in order to improve readability, the y-axis does not start at zero. The input data has been rearranged.

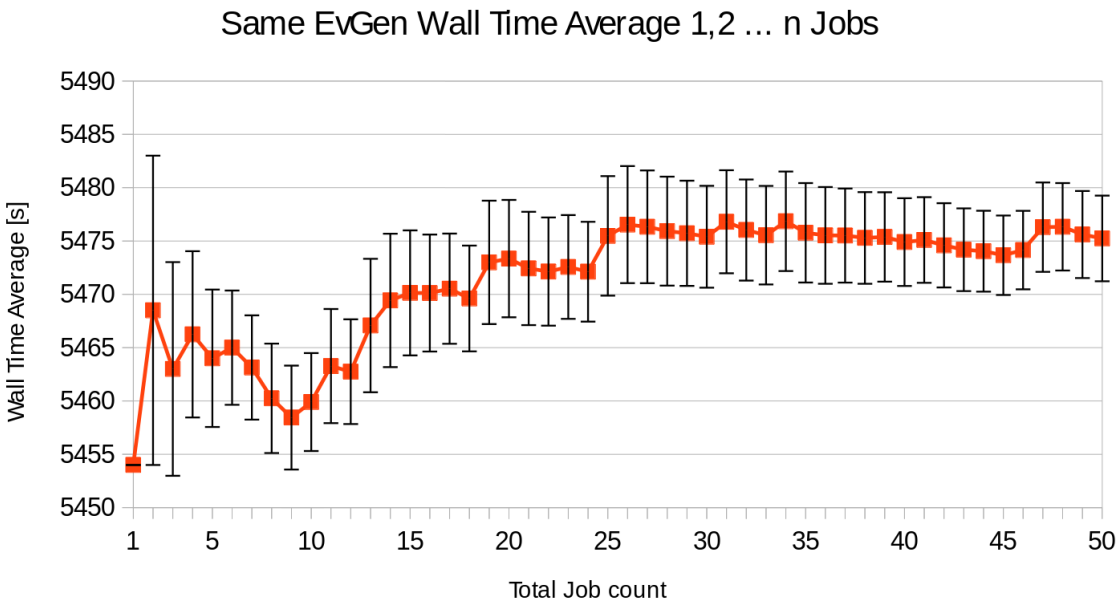

Figure A3: The wall time average over an incrementing number of same event generation jobs (starting at one), at Göttingen. The black error bars show the standard error. Note: in order to improve readability, the y-axis does not start at zero. The input data has been rearranged.

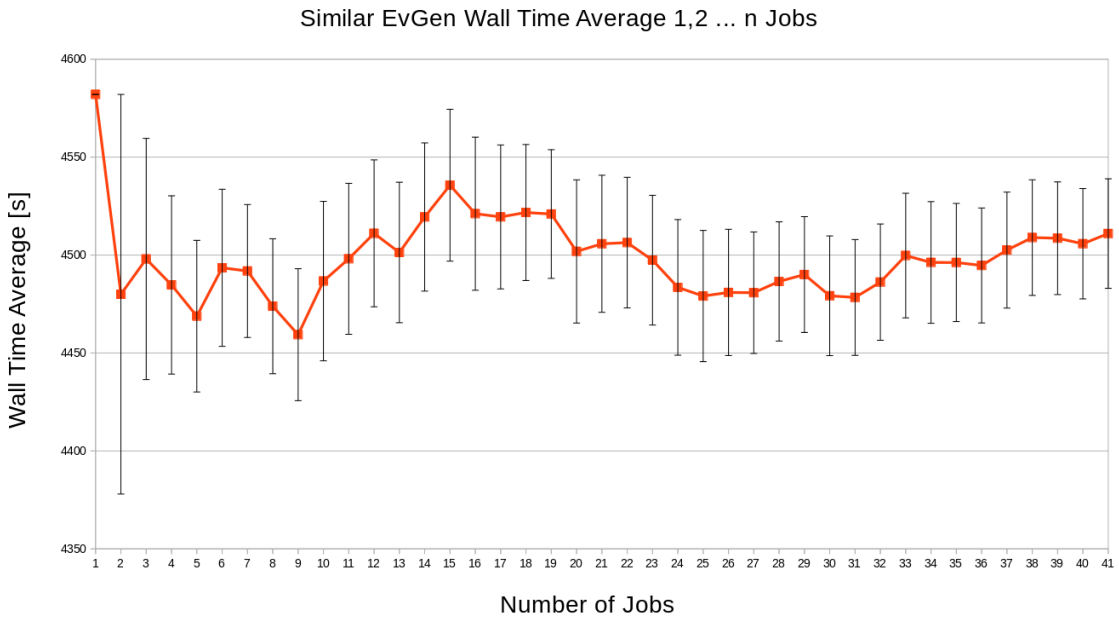

Figure A4: The wall time average over an incrementing number of *similar* event generation jobs, starting at one, at the VM at CERN. The black error bars show the standard error. Note: in order to improve readability, the y-axis does not start at zero.

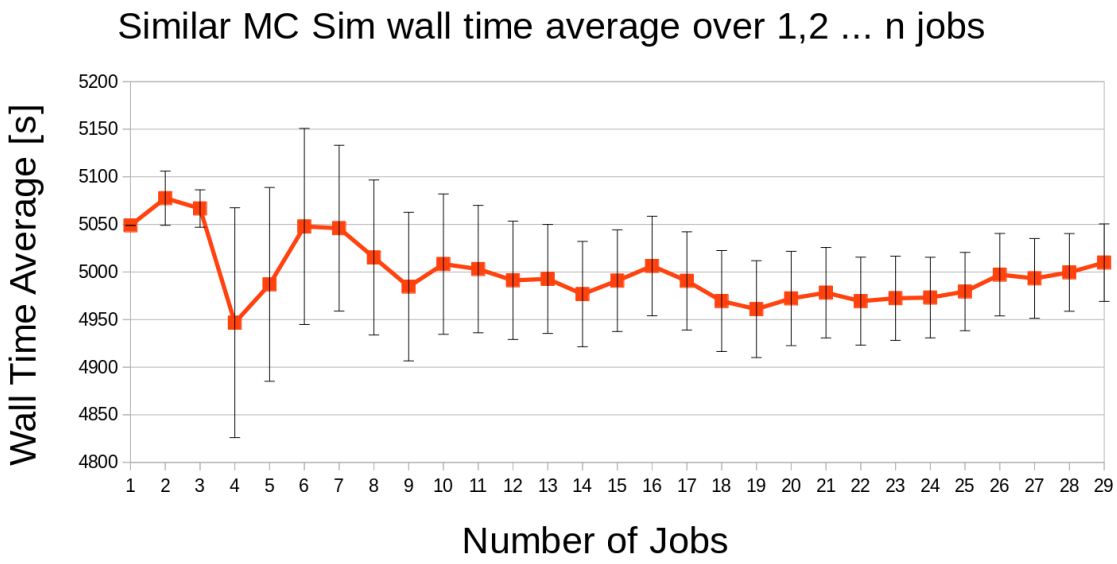

Figure A5: The wall time average over an incrementing number of *similar* Monte-Carlo simulation jobs, starting at one, at the VM at CERN. The black error bars show the standard error. Note: in order to improve readability, the y-axis does not start at zero.

Figure A6: The wall time average over an incrementing number of *similar* raw data reconstruction jobs, starting at one, at the VM at CERN. The black error bars show the standard error. Note: in order to improve readability, the y-axis does not start at zero.

A.1.2 Workflows: Raw data reconstruction

Memory limitations

The job that was executed multiple times with restricted memory was invoked with this command.

```
'/cvmfs/atlas.cern.ch/repo/sw/software/x86_64-slc6-gcc49-opt/20.7.6/Atl⌋
 ↪ asProduction/20.7.6.7/
InstallArea/share/bin/Reco_tf.py'
 '--inputBSFile=data16_13TeV.00303832.physics_Main.daq.RAW._lb0059._SFO⌋
 ↪ -6._0001.data'
 '--athenaopts=--pmon=sdmon'
 '--athenaMPStrategy' 'SharedQueue' '--maxEvents' '-1' '--postExec'
 'e2d:from AthenaCommon.AppMgr import ServiceMgr;import
 ↪ MuonRPC_Cabling.MuonRPC_CablingConfig;
ServiceMgr.MuonRPC_CablingSvc.RPCMapfromCool=False;
ServiceMgr.MuonRPC_CablingSvc.CorrFileName="LVL1confAtlasRUN2_ver016.co⌋
 ↪ rr";
ServiceMgr.MuonRPC_CablingSvc.ConfFileName="LVL1confAtlasRUN2_ver016.da⌋
 ↪ ta";'
 '--preExec'
 'all:jobproperties.Beam.bunchSpacing.set_Value_and_Lock(25);'
 'r2e:from InDetPrepRawDataToxAOD.SCTxAODJobProperties import
 ↪ SCTxAODFlags;
SCTxAODFlags.Prescale.set_Value_and_Lock(50);'
 '--autoConfiguration' 'everything' '--conditionsTag'
 ↪ 'all:CONDBR2-BLKPA-2016-15'
 '--geometryVersion'
 'all:ATLAS-R2-2015-04-00-00' '--runNumber' '303832' '--AMITag' 'f718'
 '--outputDAOD_IDTIDEFile=DAOD_IDTIDE.09066867._002142.pool.root.1'
 '--outputDESDM_CALJETFile=DESDM_CALJET.09066867._002142.pool.root.1'
 '--outputDESDM_EGAMMAFile=DESDM_EGAMMA.09066867._002142.pool.root.1'
 '--outputDESDM_EXOTHIPFile=DESDM_EXOTHIP.09066867._002142.pool.root.1'
 '--outputDESDM_MCPFile=DESDM_MCP.09066867._002142.pool.root.1'
 '--outputDESDM_PHOJETFile=DESDM_PHOJET.09066867._002142.pool.root.1'
 '--outputDESDM_SGLELFile=DESDM_SGLEL.09066867._002142.pool.root.1'
 '--outputDESDM_SLTTMUFile=DESDM_SLTTMU.09066867._002142.pool.root.1'
 '--outputDESDM_TILEMUFile=DESDM_TILEMU.09066867._002142.pool.root.1'
 '--outputDRAW_EGZFile=DRAW_EGZ.09066867._002142.pool.root.1'
 '--outputDRAW_RPVLLFile=DRAW_RPVLL.09066867._002142.pool.root.1'
 '--outputDRAW_TAUMUHFile=DRAW_TAUMUH.09066867._002142.pool.root.1'
 '--outputDRAW_ZMUMUFile=DRAW_ZMUMU.09066867._002142.pool.root.1'
 '--outputAODFile=AOD.09066867._002142.pool.root.1'
```

```
'--outputHISTFile=HIST.09066867._002142.pool.root.1' '--jobNumber'
'1603' '--ignoreErrors' 'false'
```

Swap statistic figures

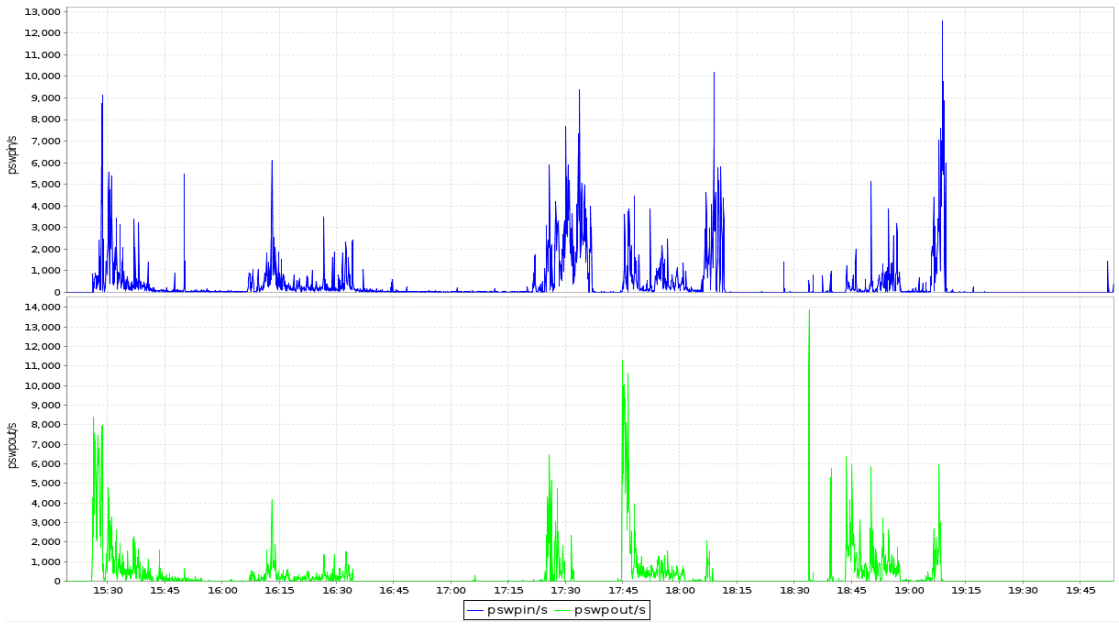

Figure A7: This plot depicts the page swap statistics for the job that had 5 GB of memory available and a wall time close to the scenario without memory limitations.

A.2 Additional profiles

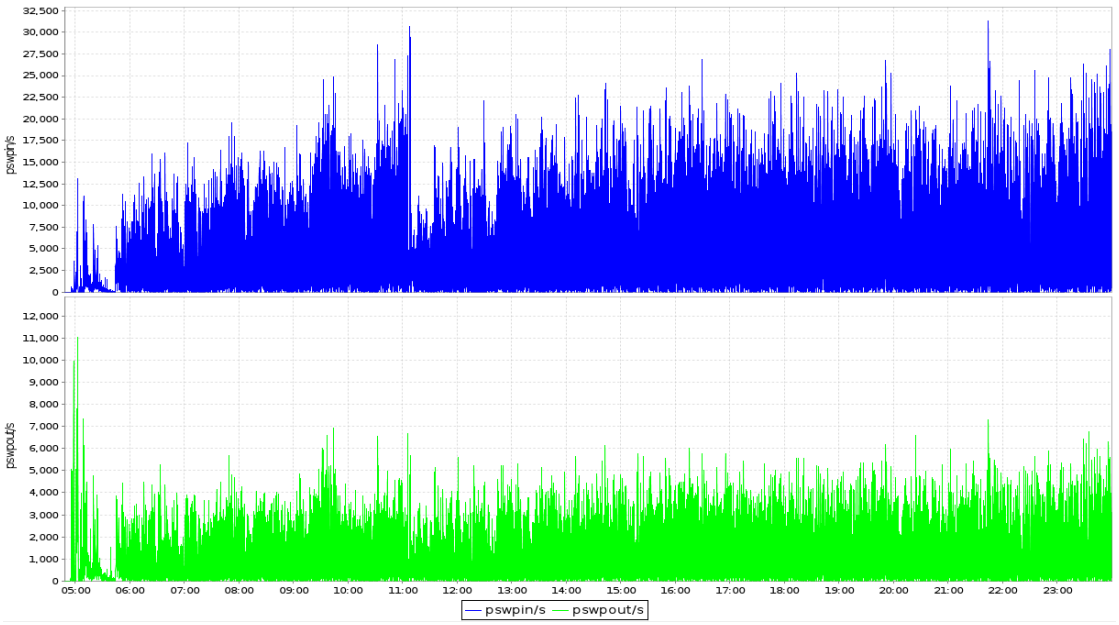

Figure A8: This plot depicts the page swap statistics for the job that had 3.75 GB of memory available and an increased wall time.

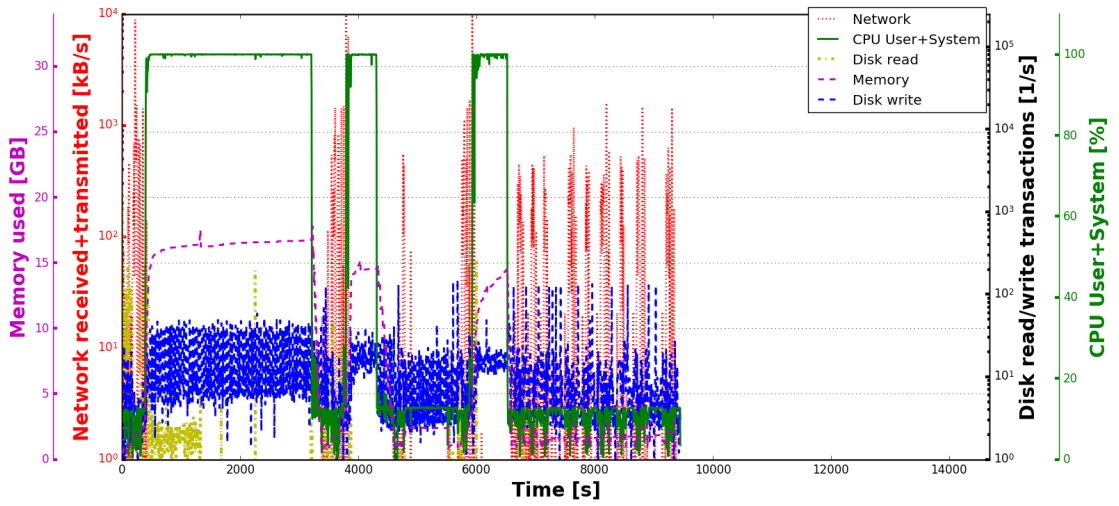

Figure A9: Normal reconstruction workflow profile (for comparison). 8 parallel AthenaMP processes on an 8 core VM.

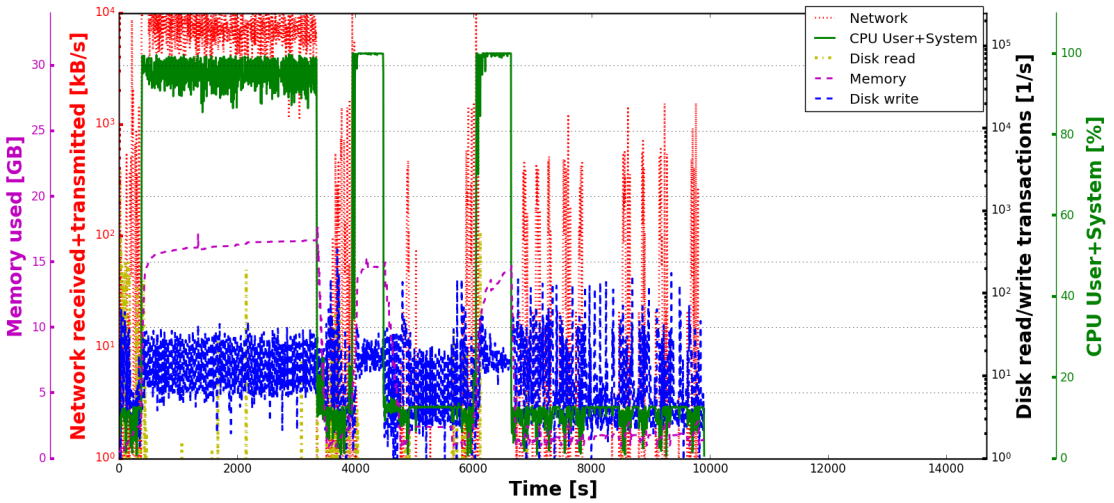

Figure A10: Reconstruction profile, showing the execution of 8 parallel AthenaMP processes on the same 8 core VM. The input data was read through the network from a remote storage at CERN.

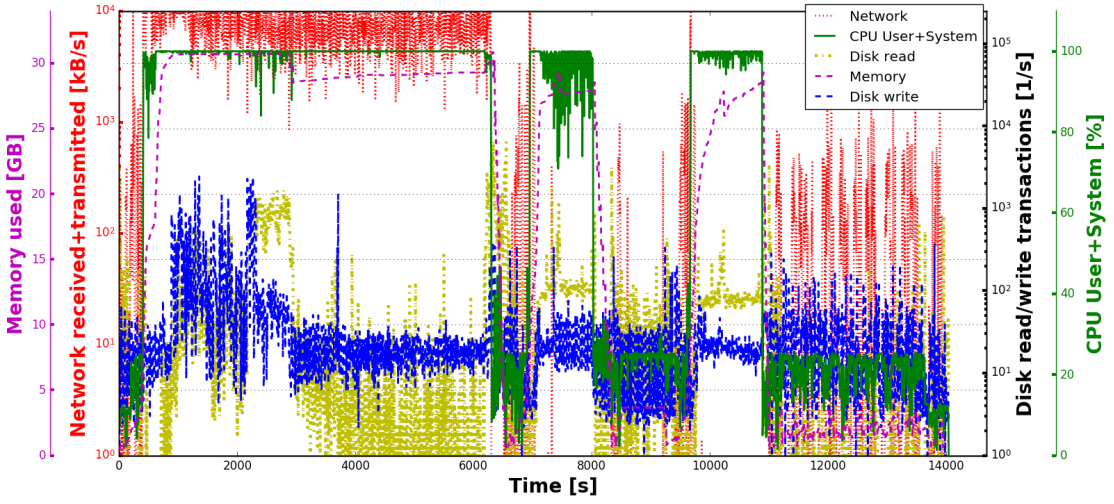

Figure A11: Overcommitted profile, showing the execution of two times 8 (16) parallel AthenaMP processes on the same 8 core VM. In addition the input data was not on the local disk. It was read through the network from a remote storage at CERN.

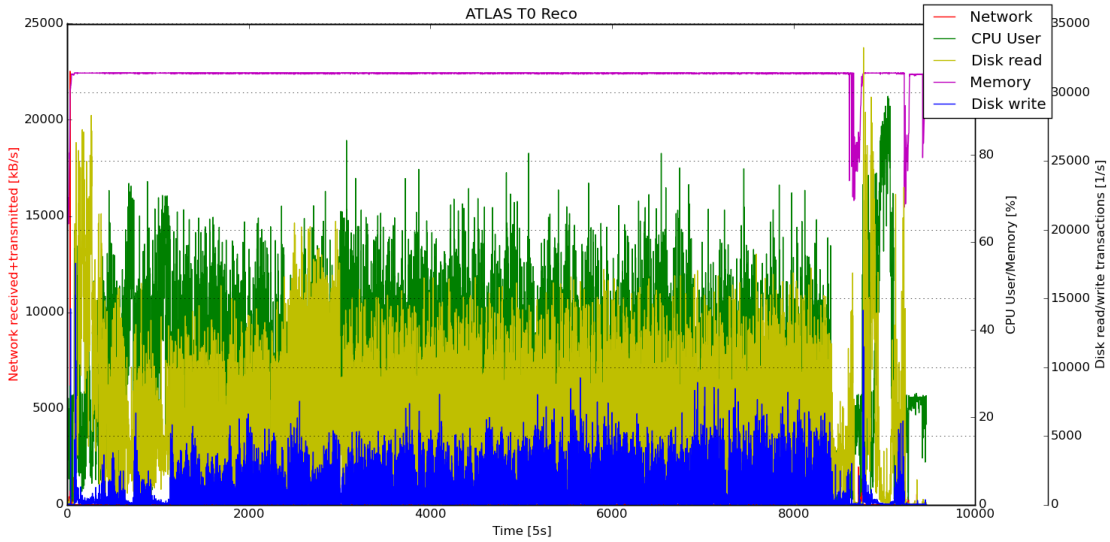

Figure A12: Job execution on a multi-core machine in which not enough RAM is available. The high disk activity stems from constant swapping between the disk and the memory, leading to a low CPU utilisation.

A.3 Model implementation

The model is computed by executing "workflow_infrastructure_model.py".

There are three separate json input files, that have to be adjusted to the use cases. The first file (input_infrastructure.json) is for the infrastructure input parameters, the second file (input_workflow.json) for the workflow input parameters and the third file (input_plot_parameters.json) for the plot parameters.

input_infrastructure.json

The infrastructure input parameters are:


```
{
    "Nr_Cores": 4.0,
    "cpu_power": 10.0,
    "io_power": 10.0,
    "idle_factor": 10.0,
    "bandwidth": 104200000,
    "RAM_machine": 8.0,
    "Storage_machine": 100.0,
    "ram_to_cpu_factor": 0.358974359,
    "bandwidth_workflow_in": 104200000,
    "bandwidth_workflow_out": 104200000,
    "disk_throughput_read": 0,
    "disk_throughput_write": 0,
    "cost_1machine_1sec": 0.0,
    "budget": 0.0,
    "infrastructure_duration": 0,
    "nr_machine_override": 1
}
```


input_workflow.json

The workflow input parameters are:


```
{
    "RAM_per_thread": 1.99,
    "Nr_Evts": 2478,
    "outbound_traffic_month": 0.0,
    "transformations": {
        "Reco_General": {
            "CPU_Idle_Time": 3.912333333,
            "CPU_Idle_Time_stdev": 1000,
            "Merge_Time": 0,
            "Validation_Time": 0,
```

```
            "Cleanup_Time": 0,
            "CPU_Time": 153.2647296,
            "CPU_Time_stdev": 1000,
            "IO_Time": 10.0,
            "IO_Time_stdev": 1000,
            "Size_Evt_In": 0,
            "Size_Evt_Out": 0,
            "is_single_core": 0,
            "position": 1,
            "OC_efficiency_gain": 0
        }
    }
}
```


input_plot_parameters.json

The plot input parameters are:


```
{
    "ram_lower_limit": 8,
    "ram_upper_limit": 9,
    "ram_nr_points": 2,
    "thread_min": 4,
    "thread_max": 5,
    "thread_stepsize": 1,
    "Z_variable": 3
}
```


They determine the x- and y-axis ranges (x-axis: from ram_lower_limit to ram_upper_limit; y-axis: from thread_min to thread_max) as well as their granularity (x-axis: ram_nr_points, the amount of points in the interval; y-axis: thread_stepsize, stepsize of points in the interval). The z_variable value corresponds to an output metric. The chosen metric will be plotted. It can be either 0: ETC; 1: EventCost; 2: EventTime; 3: Time or 4: Bandwidth.

The input variables need to keep their type (either int or float (also within lists)).

A.3.1 Model usage

After obtaining the input values and introducing them into the Model by modifying the input_infrastructure.json and the input_workflow.json file, the Model only has to be adjusted to provide the desired output metric, which is achieved by modifying the input_plot_parameters.json file. All three of these files can be found in the dedicated "input" folder.

Afterwards the Model is run by executing the ./workflow_infrastructure_model.py script.

A.3.2 Model code

The WIM is organised in the following way: the base directory contains
"Workflow_Infrastructure_Model/input
Workflow_Infrastructure_Model/modules
Workflow_Infrastructure_Model/unittest
Workflow_Infrastructure_Model/.git
Workflow_Infrastructure_Model/README.md
Workflow_Infrastructure_Model/unittest.sh
Workflow_Infrastructure_Model/workflow_infrastructure_model.py
Workflow_Infrastructure_Model/.gitignore"

workflow_infrastructure_model.py

"workflow_infrastructure_model.py" is the core executable, the contents are:

```
#!/usr/bin/python
import numpy as np
import json
from modules.plot import PLOT
from modules.combined_time_calc import combined_times
from modules.result_calc import result_calc


if __name__ == "__main__":

    #get input
    with open('input/input_infrastructure.json') as infra_input:
        infrastructure_input = json.load(infra_input)

    with open('input/input_workflow.json') as workfl_input:
        workflow_input = json.load(workfl_input)

    #read inputs
    transformation_types = []
    for value in workflow_input['transformations']:
        transformation_types.append(value)
    input_variables = {"ram_to_cpu_factor":
    ↪  infrastructure_input['ram_to_cpu_factor'], "cpu_power":
    ↪  infrastructure_input['cpu_power'],
```

```
                    "RAM_machine":
                    ↪  infrastructure_input['RAM_machine'],
                    ↪  "Storage_machine":
                    ↪  infrastructure_input['Storage_machine'],
                    "outbound_traffic_month":
                    ↪  workflow_input['outbound_traffic_month'],
                    ↪  "RAM_per_thread":
                    ↪  workflow_input['RAM_per_thread'],
                    "Nr_Cores": infrastructure_input['Nr_Cores'],
                    ↪  "cost_1machine_1sec":
                    ↪  infrastructure_input['cost_1machine_1sec'],
                    "budget": infrastructure_input['budget'],
                    ↪  "io_power": infrastructure_input['io_power']
                    ↪  , "idle_factor":
                    ↪  infrastructure_input['idle_factor']}
input_variables_int = { "Nr_Evts": workflow_input['Nr_Evts'],
↪  "bandwidth_workflow_in":
↪  infrastructure_input['bandwidth_workflow_in'],
                        "bandwidth":
                        ↪  infrastructure_input['bandwidth'],
                        ↪  "bandwidth_workflow_out": infrastructur
                        ↪  e_input['bandwidth_workflow_out'],
                        "disk_throughput_read": infrastructure_inpu
                        ↪  t['disk_throughput_read'],
                        "disk_throughput_write": infrastructure_inp
                        ↪  ut['disk_throughput_write'],
                        "infrastructure_duration": infrastructure_i
                        ↪  nput['infrastructure_duration'],
                        "nr_machine_override": infrastructure_input
                        ↪  ['nr_machine_override']}

#variable to remember if there should be multiple plots - default
↪  one plot
multiplot = ["Nr_Cores", infrastructure_input['Nr_Cores'],
↪  'cpu_power', infrastructure_input['cpu_power']]

for input_var, value in input_variables.iteritems():
    #check if input makes sense: floats
    if not isinstance(value, (float, unicode)):
        #check for lists to create multiple plots
        if isinstance(value, (list, unicode)):
```

```
            if multiplot == ["Nr_Cores",
            ↪  infrastructure_input['Nr_Cores'], 'cpu_power',
            ↪  infrastructure_input['cpu_power']]:
                multiplot = []
            multiplot.append(input_var)
            multiplot.append(value)
        else:
            raise ValueError('Make sure all float-inputs are
            ↪  actually floats', input_variables)

for input_var2, value2 in input_variables_int.iteritems():
    #check if input makes sense: integers or long
    if not isinstance(value2, (int, unicode)) and not
    ↪  isinstance(value2, (long, unicode)):
        #check for lists to create multiple plots
        if isinstance(value2, (list, unicode)):
            if multiplot == ["Nr_Cores",
            ↪  infrastructure_input['Nr_Cores'], 'cpu_power',
            ↪  infrastructure_input['cpu_power']]:
                multiplot = []
            multiplot.append(input_var2)
            multiplot.append(value2)
        else:
            raise ValueError('Make sure all int-inputs are actually
            ↪  ints (or longs)', input_variables_int)

#combine inputs into input_variables
input_variables.update(input_variables_int)

#catch case where there is no input list (only single entries)
iterator = (multiplot[1],) if not isinstance(multiplot[1], (tuple,
↪  list)) else multiplot[1]
if len(multiplot) == 2:
    multiplot.append('cpu_power')
    multiplot.append(infrastructure_input['cpu_power'])
iterator2 = (multiplot[3],) if not isinstance(multiplot[3], (tuple,
↪  list)) else multiplot[3]
#loop over input list
for variable_vary in iterator:
    #loop over second input list
    for variable_vary2 in iterator2:

        input_variables[multiplot[0]] = variable_vary
```

```
input_variables[multiplot[2]] = variable_vary2
#parse results to plot
#all z values to plot
FINAL_RESULT = []
#maximum z point x/y/z coordinates
max_result_ram = 0.0
max_result_threads = 0.0
max_result_z = 0.0
#standard (8 threads 16 GB RAM) point x/y/z coordinates
standard_z = 0.0
standard_RAM = 0.0
standard_threads = 0.0

#plot ranges, stepsize
with open('input/input_plot_parameters.json') as plot_data:
    data = json.load(plot_data)
    ram_lower_limit = data['ram_lower_limit']
    ram_upper_limit = data['ram_upper_limit']
    ram_nr_points = data['ram_nr_points']
    #create multiple plots with varying nr threads:
    thread_min = data['thread_min']
    thread_max = data['thread_max']
    thread_stepsize = data['thread_stepsize']
    #what to plot: [name, unit, function]
    Z_variable_index = data['Z_variable']

#loop over y-axis (threads)
for counter, nr_threads in enumerate(np.arange(thread_min,
↪  thread_max, thread_stepsize)):
    result11 = np.linspace(ram_lower_limit,ram_upper_limit,
    ↪  ram_nr_points)
    #loop over x-axis (ram)
    for counter1, amount_RAM in
    ↪  enumerate(np.linspace(ram_lower_limit,
    ↪  ram_upper_limit, ram_nr_points)):
        processing_time_l = []
        downloadtime_l = 0
        inputsize_l = 0
        CPU_Time_l = 0
        #loop over transformations within a workflow
        for transformation in transformation_types:
            #calculate durations
```

```
combined_time = combined_times(nr_threads,
↪  transformation, amount_RAM,
                processing_time_t = workflow_in
                ↪  put["transformations"][tran
                ↪  sformation]["CPU_Time"],
                Merge_Time_t = workflow_input["
                ↪  transformations"][transform
                ↪  ation]["Merge_Time"],
                cpu_idle_t = workflow_input["tr
                ↪  ansformations"][transformat
                ↪  ion]["CPU_Idle_Time"],
                validation_t = workflow_input["
                ↪  transformations"][transform
                ↪  ation]["Validation_Time"],
                cleanup_t = workflow_input["tra
                ↪  nsformations"][transformati
                ↪  on]["Cleanup_Time"],
                Size_Evt_Out = workflow_input["
                ↪  transformations"][transform
                ↪  ation]["Size_Evt_Out"],
                Size_Evt_In = workflow_input["t
                ↪  ransformations"][transforma
                ↪  tion]["Size_Evt_In"],
                singlecore_t = workflow_input["
                ↪  transformations"][transform
                ↪  ation]["is_single_core"],
                Nr_Cores = input_variables['Nr_
                ↪  Cores'], cpu_power =
                ↪  input_variables['cpu_power'
                ↪  ],
                RAM_per_thread = input_variable
                ↪  s['RAM_per_thread'],
                ↪  Nr_Evts =
                ↪  input_variables['Nr_Evts'],
                bandwidth_workflow_out =
                ↪  input_variables['bandwidth_
                ↪  workflow_out'],
                bandwidth_workflow_in =
                ↪  input_variables['bandwidth_
                ↪  workflow_in'],
                disk_throughput_read =
                ↪  input_variables['disk_throu
                ↪  ghput_read'],
```

```
                                disk_throughput_write =
                                ↪  input_variables['disk_throu
                                ↪  ghput_write'],
                                position_t = workflow_input["tr
                                ↪  ansformations"][transformat
                                ↪  ion]["position"],
                                OC_efficiency_gain_t =
                                ↪  workflow_input["transformat
                                ↪  ions"][transformation]['OC_
                                ↪  efficiency_gain'],
                                io_power =
                                ↪  input_variables['io_power'],
                                idle_factor = input_variables['
                                ↪  idle_factor'],
                                IO_time_t = workflow_input["tra
                                ↪  nsformations"][transformati
                                ↪  on]["IO_Time"])
            processing_time_l.append(combined_time.total_pr
            ↪  ocessing_time(Nr_Cores =
            ↪  input_variables['Nr_Cores'],
                                    Nr_Evts = input_variab
                                    ↪  les['Nr_Evts']))
            downloadtime_l = downloadtime_l +
            ↪  combined_time.stagein
            inputsize_l = inputsize_l +
            ↪  combined_time.inputsize
            CPU_Time_l = CPU_Time_l +
            ↪  combined_time.CPU_Time_total
            total_processing_time = sum(processing_time_l)
        #final result
        result = result_calc(total_processing_time=total_pr
        ↪  ocessing_time, nr_threads=nr_threads,
        ↪  transformation=transformation,
                 RAM_j=amount_RAM,
                 ↪  Nr_Cores=input_variables['Nr_Cores'],
                 RAM_machine=input_variables['RAM_machine']
                 ↪  ,
                 ↪  ram_to_cpu_factor=input_variables['ram
                 ↪  _to_cpu_factor'],
                 input_size = workflow_input["transformatio
                 ↪  ns"][transformation]["Size_Evt_In"],
```

```
            budget=input_variables['budget'],
            ↪  cost_1machine_1sec=input_variables['co
            ↪  st_1machine_1sec'],
            infrastructure_duration=input_variables['i
            ↪  nfrastructure_duration'],
            ↪  nr_machine_override=input_variables['n
            ↪  r_machine_override'],
            CPU_Idle_Time_stdev =
            ↪  workflow_input["transformations"][tran
            ↪  sformation]["CPU_Idle_Time_stdev"],
            CPU_Time_stdev =
            ↪  workflow_input["transformations"][tran
            ↪  sformation]["CPU_Time_stdev"],
            IO_Time_stdev =
            ↪  workflow_input["transformations"][tran
            ↪  sformation]["IO_Time_stdev"],
            Nr_Evts = input_variables['Nr_Evts'])
    #list of possible results chosen by index
    Z_variable = [["ETC", "Events/second/CHF",
    ↪  result.ETC_calc, result.ETC_calc_unc],
                  ["EventCost", "Event/CHF",
                  ↪  result.EventCost_calc,
                  ↪  result.EventCost_calc_unc],
                  ["EventTime", "Events/second",
                  ↪  result.EventTime_calc,
                  ↪  result.EventTime_calc_unc],
                  ["Time", "seconds", result.Time_calc,
                  ↪  result.Time_all_calc_unc],
                  ["Bandwidth", "Gb/second",
                  ↪  result.Bandwidth_calc,
                  ↪  result.Bandwidth_calc_unc],
                  ["CostEvent", "CHF/Event",
                  ↪  result.CostEvent_calc,
                  ↪  result.CostEvent_calc_unc],
                  ["Events", "Events",
                  ↪  result.Events_calc,
                  ↪  result.Events_calc_unc],
                  ["Downloadspeed", "Bytes/second",
                  ↪  result.Input_bytes_second_calc, re
                  ↪  sult.Input_bytes_second_calc_unc],
                  ["InputSize", "Bytes",
                  ↪  result.Input_bytes_calc,
                  ↪  result.Input_bytes_calc_unc],
```

```
                ["DownloadTime", "seconds",
                ↪  result.DownloadTime_calc,
                ↪  result.DownloadTime_calc_unc],
                ["CPUTime", "seconds",
                ↪  result.CPUTime_calc,
                ↪  result.CPUTime_calc_unc]]

            if (Z_variable_index == 7) or (Z_variable_index ==
            ↪  8) or (Z_variable_index == 9):
                chosen_result = Z_variable[Z_variable_index][2]
                ↪  (downloadtime_l,
                ↪  inputsize_l)
                Total_Uncertainty =
                ↪  Z_variable[Z_variable_index][3]()

            elif (Z_variable_index == 10):
                chosen_result =
                ↪  Z_variable[Z_variable_index][2](CPU_Time_l,
                ↪  Nr_Evts = input_variables['Nr_Evts'])
                Total_Uncertainty =
                ↪  Z_variable[Z_variable_index][3]()

            else:
                #duration of one full workflow (sum of all
                ↪  transformations)
                chosen_result = Z_variable[Z_variable_index][2]
                ↪  (total_processing_time)
                Total_Uncertainty =
                ↪  Z_variable[Z_variable_index][3]()

            if result.nr_machines != 0:
                #make sure input bandwidthes agree with each
                ↪  other: factor of 8 to account for downloads
                ↪  not in parallel
                if (input_variables['bandwidth'] /
                ↪  result.nr_machines * 8) <
                ↪  (input_variables['bandwidth_workflow_in']):
                    raise ValueError('Input disagreement:
                    ↪  bandwidth not possible with
                    ↪  bandwidth_workflow_in')
```

```
                if (input_variables['bandwidth'] /
                ↪  result.nr_machines * 8) <
                ↪  (input_variables['bandwidth_workflow_out']):
                    raise ValueError('Input disagreement:
                    ↪  bandwidth not possible with
                    ↪  bandwidth_workflow_out')
            else:
                continue
            #find maximum
            if chosen_result > max_result_z:
                max_result_z = chosen_result
                max_result_ram = amount_RAM
                max_result_threads = nr_threads
                max_Total_Uncertainty = Total_Uncertainty
            #find standard point
            if abs(nr_threads-input_variables['Nr_Cores']) <=
            ↪  abs(standard_threads-input_variables['Nr_Cores'
            ↪  ]):
                if abs(amount_RAM-input_variables['Nr_Cores']*2
                ↪  ) <=
                ↪  abs(standard_RAM-input_variables['Nr_Cores'
                ↪  ]*2):
                    standard_z = chosen_result
                    standard_threads = nr_threads
                    standard_RAM = amount_RAM
                    standard_Total_Uncertainty =
                    ↪  Total_Uncertainty
            #put result in plottable format
            result11[counter1] = chosen_result
        FINAL_RESULT.append(result11)
    #create + save plot
    PLOT(ram_lower_limit, ram_upper_limit, ram_nr_points,
    ↪  thread_min, thread_max, thread_stepsize,
    ↪  max_result_ram, max_result_threads,
         max_result_z, standard_RAM, standard_threads,
         ↪  standard_z, FINAL_RESULT, multiplot,
         ↪  variable_vary, Z_variable[Z_variable_index],
         variable_vary2, max_Total_Uncertainty,
         ↪  standard_Total_Uncertainty)
```

Workflow_Infrastructure_Model/input

The subdirectory ‘input’ contains the files:
“input_infrastructure.json input_plot_parameters.json input_workflow.json”

The contents of these files have been shown in the Appendix, see subsection A.3.

Workflow_Infrastructure_Model/modules

The subdirectory 'modules' contains the files:
"combined_time_calc.py efficiency_calculation.py plot.py transformation_time_calc.py cost_calculation.py __init__.py result_calc.py uncertainty_estimation.py" These modules form the core logic of the model. They combine the different input parameters to intermediate and final results.

combined_time_calc.py

```
from math import sqrt
from transformation_time_calc import transformation_time
from efficiency_calculation import overcommit_efficiency

class combined_times(transformation_time, overcommit_efficiency):
    "combining the times"
    def __init__(self, nr_threads, transformation, RAM_j,
    ↪  processing_time_t, Merge_Time_t, Size_Evt_In, Size_Evt_Out,
                cpu_idle_t, validation_t, cleanup_t, singlecore_t,
                ↪  Nr_Cores, cpu_power, RAM_per_thread, Nr_Evts,
                ↪  OC_efficiency_gain_t,
                bandwidth_workflow_in, bandwidth_workflow_out,
                ↪  position_t, disk_throughput_read,
                ↪  disk_throughput_write, io_power, idle_factor,
                ↪  IO_time_t):
        transformation_time.__init__(self, nr_threads, transformation,
        ↪  RAM_j, processing_time_t, Merge_Time_t,
                                    bandwidth_workflow_in, cpu_idle_t,
                                    ↪  validation_t, cleanup_t,
                                    ↪  bandwidth_workflow_out,
                                    ↪  singlecore_t,
                                    Nr_Cores, cpu_power,
                                    ↪  RAM_per_thread, Size_Evt_In,
                                    ↪  Size_Evt_Out, Nr_Evts,
                                    ↪  position_t,
                                    disk_throughput_read,
                                    ↪  disk_throughput_write,
                                    ↪  io_power, idle_factor,
                                    ↪  IO_time_t)
        overcommit_efficiency.__init__(self, nr_threads, Nr_Cores,
        ↪  OC_efficiency_gain_t)
```

```
        self.workflow_time = self.workflow_time_calculate(Nr_Evts)
        self.idle_time = self.idle_time_uc_calculate(Nr_Cores)
        self.cpu_impact, self.idle_impact, self.IO_impact =
        ↪  self.component_impact_workflow(Nr_Evts)
        self.total_error = self.fixed_error(Nr_Evts)

    def workflow_time_calculate(self, Nr_Evts):
        "calculate workflow duration (sum all cores)"
        workflow_time = (self.CPU_Time_total  * Nr_Evts +
        ↪  (self.IO_Time_total + self.Swap_Time_total +
        ↪  self.Merge_Time_total) * self.nr_threads +
                        self.Const_overhead_Time_total +
                        ↪  self.Singlecore_Idle_total * Nr_Evts)
        return workflow_time

    def component_impact_workflow(self, Nr_Evts):
        "calculate the impact of cpu time, idle time and IO time on the
        ↪  overall wall time in percent"
        cpu_impact = self.CPU_Time_total  * Nr_Evts /
        ↪  self.workflow_time * 100
        idle_impact = (self.Const_overhead_Time_total +
        ↪  self.Singlecore_Idle_total * Nr_Evts) / self.workflow_time
        ↪  * 100
        IO_impact = (self.IO_Time_total + self.Swap_Time_total +
        ↪  self.Merge_Time_total) * self.nr_threads /
        ↪  self.workflow_time * 100
        return cpu_impact, idle_impact, IO_impact

    def fixed_error(self, Nr_Evts):
        "the error is estimated by the maximum deviation found from
        ↪  many measurements of 'similar jobs' reconstruction. Then
        ↪  error propagation is applied."
        error_cpu = self.CPU_Time_total  * Nr_Evts * 0.05
        error_idle = (self.Const_overhead_Time_total +
        ↪  self.Singlecore_Idle_total * Nr_Evts) * 0.13

        if (self.IO_Time_total + self.Swap_Time_total +
        ↪  self.Merge_Time_total) * self.nr_threads /
        ↪  self.workflow_time < 0.05:
            error_io = (self.IO_Time_total + self.Swap_Time_total +
            ↪  self.Merge_Time_total) * self.nr_threads * 0.66
        else:
            error_io = 1000
```

```
        error_total = sqrt(error_cpu * error_cpu + error_idle *
        ↪  error_idle + error_io * error_io)
        return error_total

    def idle_time_uc_calculate(self, Nr_Cores):
        "calculate idle time when under-committing - basically the job
        ↪  duration multiplied by number of idle cores"
        if Nr_Cores > self.nr_threads:
            idle_time = self.workflow_time / self.nr_threads *
            ↪  (Nr_Cores - self.nr_threads)
        else:
            idle_time = 0
        return idle_time

    def overhead_calculate(self, Nr_Cores, Nr_Evts):
        "calculate overhead that can be subtracted"
        overhead_time = self.OCF * (self.nr_threads - Nr_Cores) /
        ↪  self.nr_threads * (self.CPU_Time_total  * Nr_Evts +
                        self.IO_Time_total + self.Swap_Time_total +
                        ↪  self.Merge_Time_total +
                        ↪  self.Const_overhead_Time_total)
        return overhead_time

    def total_processing_time(self, Nr_Cores, Nr_Evts):
        "calculate processing duration"
        processing_duration = self.workflow_time -
        ↪  self.overhead_calculate(Nr_Cores, Nr_Evts) + self.idle_time
        return processing_duration
```

efficiency_calculation.py

```
class overcommit_efficiency:
    "workflow specific factor"
    def __init__(self, nr_threads, Nr_Cores, OC_efficiency_gain):
        self.overcommit_factor = nr_threads / Nr_Cores
        self.OCF =
        ↪  self.calculate_overcommit_efficiency(OC_efficiency_gain)

    def calculate_overcommit_efficiency(self, OC_efficiency_gain):
        """calculate overcommit efficiency (only if unset meaning
        ↪  zero)... meaning the percentage of overcommitted time that
        ↪  doesn't influence the job duration,
           because it uses (previous) CPU idle time"""
```

```
        if OC_efficiency_gain == 0:
            if self.overcommit_factor > 1:
                if self.overcommit_factor > 2:
                    self.OCF = 0.7 / (2**(self.overcommit_factor-2))
                else:
                    self.OCF = 0.05 * self.overcommit_factor + 0.6
            else:
                self.OCF = 0
        return self.OCF
```

plot.py

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from mpl_toolkits.mplot3d import Axes3D
from math import log10, floor
class PLOT():
    "plot results"
    def __init__(self, ram_lower_limit, ram_upper_limit, ram_nr_points,
    ↪  thread_min, thread_max, thread_stepsize, max_result_ram,
    ↪  max_result_threads,
                 max_result_z, standard_RAM, standard_threads,
                 ↪  standard_z, FINAL_RESULT, multiplot,
                 ↪  variable_vary, Z_variable, variable_vary2,
                 max_Total_Uncertainty, standard_Total_Uncertainty):
        x = np.linspace(ram_lower_limit,ram_upper_limit,ram_nr_points)
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.set_xlabel('RAM [GB]', color='b', fontsize=20)
        ax.set_zlabel(Z_variable[0] + ' ' + Z_variable[1], color='b',
        ↪  fontsize=20)
        ax.set_ylabel('Number of processes', color='b', fontsize=20)

        ax.set_xlim([ram_lower_limit-3,ram_upper_limit+3])
        ax.set_ylim([thread_min-3,thread_max+1])
        if Z_variable[0] == 'Time':
            ax.set_zlim([0,30000])
        ax.xaxis.labelpad = 12
        ax.yaxis.labelpad = 10
        ax.zaxis.labelpad = 10

        thread_list = np.arange(thread_min, thread_max, thread_stepsize)
        X,Y = np.meshgrid(x,thread_list)
```

```
        Z = FINAL_RESULT
        ax.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.9,
        ↪  cmap=plt.cm.jet, linewidth=0)

        ax.scatter([max_result_ram], [max_result_threads],
        ↪  [max_result_z], 'ok', s = 150)
        ax.text(max_result_ram, max_result_threads,
        ↪  max_result_z+0.0001, self.round_to_1(max_result_z), size =
        ↪  20)
        ax.scatter([standard_RAM], [standard_threads], [standard_z],
        ↪  'ok')


        cset = ax.contour(X, Y, Z, zdir='z', offset=-1,
        ↪  cmap=cm.coolwarm)
        cset = ax.contour(X, Y, Z, zdir='x', offset=+3,
        ↪  cmap=cm.coolwarm)
        cset = ax.contour(X, Y, Z, zdir='y', offset=+3,
        ↪  cmap=cm.coolwarm)

        print 'Maximum:  RAM: ', max_result_ram, 'Threads: ',
        ↪  max_result_threads, Z_variable[0] + ': ', max_result_z,
        ↪  '+-',  max_Total_Uncertainty,\
              ' ['+ Z_variable[1] + ']', ', Error [%] = ',
              ↪  (max_Total_Uncertainty / max_result_z * 100)
        print 'Standard:  RAM: ', standard_RAM, 'Threads: ',
        ↪  standard_threads, Z_variable[0]  + ': ', standard_z, '+-',
        ↪  standard_Total_Uncertainty,\
              ' ['+ Z_variable[1] + ']' , ', Error [%] = ',
              ↪  (standard_Total_Uncertainty / standard_z * 100)

        ax.legend()
        ax.azim=45
        ax.elev=08.
        fig.savefig(multiplot[0] +'_' + str(variable_vary) + '_' +
        ↪  multiplot[2] +'_'+ str(variable_vary2) + '_'
                    + str(Z_variable[0]) + '.jpg', bbox_inches='tight')
        plt.show()
        plt.close()
    def round_to_1(self, x):
        if x == 0.0:
            return x
```

```
        elif not isinstance(x, (int, unicode)) and not isinstance(x,
        ↪  (long, unicode)):
            return round(x, -int(floor(log10(abs(x)))-1))
        else:
            return int(round(x))
```

transformation_time_calc.py

```
class transformation_time:
    "calculate all relevant durations: swap, processing and constant
    ↪  time"
    def __init__(self, nr_threads, transformation, RAM_j,
    ↪  processing_time_t, Merge_Time_t,
                bandwidth_workflow_in, cpu_idle_t, validation_t,
                ↪  cleanup_t, bandwidth_workflow_out, singlecore_t,
                ↪  Nr_Cores, cpu_power,
                RAM_per_thread, Size_Evt_In, Size_Evt_Out, Nr_Evts,
                ↪  position_t, disk_throughput_read,
                ↪  disk_throughput_write, io_power, idle_factor,
                ↪  IO_time_t):
        self.nr_threads = nr_threads
        self.Nr_Cores = Nr_Cores
        self.transformation = transformation
        self.RAM_j = RAM_j
        self.overcommit_factor = self.nr_threads / Nr_Cores
        self.processing_time = processing_time_t
        self.idle_cpu_time = cpu_idle_t
        self.validation = validation_t
        self.cleanup = cleanup_t
        self.singlecore = singlecore_t
        self.position = position_t
        self.stagein, self.inputsize =
        ↪  self.Network_Read_Time(bandwidth_workflow_in, Size_Evt_In,
        ↪  Nr_Evts)
        self.stageout = self.Network_Write_Time(bandwidth_workflow_out,
        ↪  Size_Evt_Out, Nr_Evts)
        self.CPU_Time_total = self.CPU_Time(cpu_power, Nr_Evts)
        self.IO_Time_total = self.IO_Time(disk_throughput_read,
        ↪  disk_throughput_write, Size_Evt_In, Size_Evt_Out, Nr_Evts,
        ↪  io_power, IO_time_t)
        self.Idle_Time_total = self.Idle_Time(idle_factor, Nr_Evts)
        self.Swap_Time_total = self.Swap_Time(RAM_per_thread)
        self.Const_overhead_Time_total =
        ↪  self.Const_overhead_Time(cpu_power, Nr_Evts)
```

```
        self.Merge_Time_total = self.Merge_Time(Merge_Time_t)
        self.Singlecore_Idle_total = self.CPU_Singlecore_Idle_Time()

    def Network_Read_Time(self, bandwidth_workflow_in, Size_Evt_In,
    ↪  Nr_Evts):
        "return the stage-in duration(network) and disk read duration"
        if self.position == 0:
            stagein_t = Size_Evt_In * Nr_Evts / bandwidth_workflow_in
            inputsize = Size_Evt_In * Nr_Evts
        else:
            stagein_t = 0
            inputsize = 0
        return stagein_t, inputsize

    def Network_Write_Time(self, bandwidth_workflow_out, Size_Evt_Out,
    ↪  Nr_Evts):
        "return the stage-out duration(network)"
        if self.position == 2:
            stageout_t = Size_Evt_Out * Nr_Evts / bandwidth_workflow_out
        else:
            stageout_t = 0
        return stageout_t

    def CPU_Time(self, cpu_power, Nr_Evts):
        "return total CPU Time"
        #single core means idle cores that do nothing
        if self.singlecore:
            CPU_Time1 = self.processing_time / cpu_power
        else:
            CPU_Time1 = self.processing_time / cpu_power *
            ↪  self.nr_threads / self.Nr_Cores
        return CPU_Time1

    def IO_Time(self, disk_throughput_read, disk_throughput_write,
    ↪  Size_Evt_In, Size_Evt_Out, Nr_Evts, io_power, IO_time_t):
        "return I/O time"
        IO_Time_total = IO_time_t / io_power / self.Nr_Cores * Nr_Evts
        return IO_Time_total

    def Idle_Time(self, idle_factor, Nr_Evts):
        "return Idle time"
        Idle_time_res = self.idle_cpu_time / idle_factor * 10 * Nr_Evts
        return Idle_time_res
```

```
def swap_penalty(self):
    "calculate the swap overhead time given a RAM value...
    ↪  simplification: penalised heavily by constant"
    #only for transfomrations that actually swap
    if "RAWtoESD" in self.transformation:
        swap_overhead = 100000000000000
    else:
        swap_overhead=0
    return swap_overhead

def Swap_Time(self, RAM_per_thread):
    "calculate swap time through penalty function (applied after
    ↪  RAM per Core ratio becomes too low)"
    penalty_lim = RAM_per_thread * self.nr_threads #after this
    ↪  swapping-penalty (runtime):
    if self.RAM_j < penalty_lim:
        if self.swap_penalty() > 0:
            Swap_Time_1 = self.swap_penalty()
        else:
            Swap_Time_1 = 0
    else:
        Swap_Time_1 = 0
    return Swap_Time_1

def Merge_Time(self, Merge_Time_t):
    return Merge_Time_t*self.nr_threads

def CPU_Singlecore_Idle_Time(self):
    "if (serial step) merging happens, add idle time"
    CPU_idle_singlecore = 0
    if self.singlecore:
        CPU_idle_singlecore = self.CPU_Time_total *
        ↪  (self.nr_threads - 1)
    else:
        CPU_idle_singlecore = 0
    return CPU_idle_singlecore

def Const_overhead_Time(self, cpu_power, Nr_Evts):
    "sum over all the constant times: startup, setup etc..."
    self.Const_overhead_Time_total = (self.stagein +
    ↪  self.Idle_Time_total + self.validation + self.cleanup +
    ↪  self.stageout) * self.nr_threads
```

```
        return self.Const_overhead_Time_total
```

cost_calculation.py

```
class cost_calculation:
    "calculate cost of infrastrucutre"
    def __init__(self, RAM_i, Nr_Cores, RAM_machine, ram_to_cpu_factor,
    ↪  budget, cost_1machine_1sec, infrastructure_duration,
    ↪  nr_machine_override):
        "create an instance for every RAM amount"
        self.RAM_i = RAM_i
        self.Nr_Cores = Nr_Cores
        self.RAM_machine = RAM_machine
        self.ram_to_cpu_factor = ram_to_cpu_factor
        self.cost_1machine_1sec = cost_1machine_1sec
        self.infrastructure_duration = infrastructure_duration
        self.machines = self.budget_machines(budget,
        ↪  nr_machine_override)
        self.nr_machines = self.total_machines(budget,
        ↪  nr_machine_override)
        self.total_cost_machines = budget

    def budget_machines(self, budget, nr_machine_override):
        "amount of standard machines the budget allows for"
        if nr_machine_override == 0:
            if (budget / self.cost_1machine_1sec *
            ↪  self.infrastructure_duration) >= 1:
                return int(budget / (self.cost_1machine_1sec *
                ↪  self.infrastructure_duration))
            else:
                raise ValueError('Input disagreement: budget cannot
                ↪  afford machines')

    def total_machines(self, budget, nr_machine_override):
        "return number of machines, considering RAM variation.
        ↪  nr_machine_override if non-zero overrides all budget, ram
        ↪  and core considerations"
        if nr_machine_override != 0:
            return nr_machine_override
        else:
            if self.Nr_Cores <= 0:
                nr_machines_1 = 0
            else:
```

```
                nr_machines_1 = (self.Nr_Cores - (self.RAM_i -
                ↪  self.RAM_machine) * self.ram_to_cpu_factor) *
                ↪  self.machines / self.Nr_Cores
            if nr_machines_1 >= 0:
                return nr_machines_1
            else:
                return 0
```

result_calc.py

```
from cost_calculation import cost_calculation
from uncertainty_estimation import estimate_uncertainty


class result_calc(cost_calculation, estimate_uncertainty):
    "calculate the chosen result"
    def __init__(self, total_processing_time, nr_threads,
    ↪  transformation, RAM_j, Nr_Cores, RAM_machine,
    ↪  ram_to_cpu_factor, input_size, budget,
                 cost_1machine_1sec, infrastructure_duration,
                 ↪  nr_machine_override, CPU_Idle_Time_stdev,
                 ↪  CPU_Time_stdev, IO_Time_stdev, Nr_Evts):
        cost_calculation.__init__(self, RAM_i=RAM_j, Nr_Cores=Nr_Cores,
        ↪  RAM_machine=RAM_machine,
        ↪  ram_to_cpu_factor=ram_to_cpu_factor,
                                  budget=budget, cost_1machine_1sec=cos⌋
                                  ↪  t_1machine_1sec,
                                  ↪  infrastructure_duration=infrastru⌋
                                  ↪  cture_duration,
                                  nr_machine_override=nr_machine_overri⌋
                                  ↪  de)
        estimate_uncertainty.__init__(self, Nr_Evts)
        self.total_processing_time = total_processing_time
        self.overcommit_factor = nr_threads / Nr_Cores
        self.Nr_Cores = Nr_Cores
        self.input_size = input_size
        self.CPU_Idle_Time_stdev = CPU_Idle_Time_stdev
        self.CPU_Time_stdev = CPU_Time_stdev
        self.IO_Time_stdev = IO_Time_stdev
        self.num_workflows = self.num_workflows_calc()
        self.sum_walltimes = self.sum_walltimes_calc()
        self.Total_Events = self.total_events_calc(Nr_Evts)

    def total_events_calc(self, Nr_Evts):
```

```
        "total number of events produced"
        if self.sum_walltimes == 0:
            raise ValueError('wall time is zero')
        Total_Events = Nr_Evts * self.overcommit_factor *
        ↪  self.num_workflows
        return Total_Events

    def sum_walltimes_calc(self):
        "sum of all walltimes"
        if self.Nr_Cores == 0:
            raise ValueError('Nr_Cores input is zero')
        sum_walltimes = self.total_processing_time * self.num_workflows
        ↪  / self.Nr_Cores
        return sum_walltimes

    def num_workflows_calc(self):
        "how many workflows are run in total"
        if self.Nr_Cores == 0:
            raise ValueError('Nr_Cores input is zero')
        num_workflows = self.infrastructure_duration * self.nr_machines
        ↪  / (self.total_processing_time / self.Nr_Cores)
        return num_workflows

    def ETC_calc(self, Nr_Evts):
        "events/second/chf"
        if self.total_cost_machines == 0:
            raise ValueError('budget input is zero')
        if self.sum_walltimes == 0:
            raise ValueError('wall time is zero')
        ETC_res = self.Total_Events / self.sum_walltimes /
        ↪  self.total_cost_machines
        return ETC_res

    def EventCost_calc(self, Nr_Evts):
        "events/chf"
        if self.total_cost_machines == 0:
            raise ValueError('budget input is zero')
        EC_res = self.Total_Events / self.total_cost_machines
        return EC_res

    def EventTime_calc(self, Nr_Evts):
        "events/second"
        if self.sum_walltimes == 0:
```

```
            raise ValueError('wall time is zero')
        ET_res = self.Total_Events / self.sum_walltimes
        return ET_res

    def Time_calc(self, Nr_Evts):
        "total time"
        T_res = self.sum_walltimes
        return T_res

    def Bandwidth_calc(self, Nr_Evts):
        "bandwidth"
        Bandwidth_res = self.input_size * self.Total_Events /
        ↪  self.sum_walltimes / 1000000000 * 8
        return Bandwidth_res

    def CostEvent_calc(self, Nr_Evts):
        "chf/event"
        if self.Total_Events == 0:
            raise ValueError('total events are zero')
        CE_res = self.total_cost_machines / self.Total_Events
        return CE_res

    def Events_calc(self, Nr_Evts):
        "events"
        Events_res = self.Total_Events
        return Events_res

    def Input_bytes_second_calc(self, downloadtime_l, inputsize_l,
    ↪  Nr_Evts):
        "input size per downloadtime [bytes/second]"
        if downloadtime_l == 0:
            raise ValueError('downloadtime input is zero')
        if inputsize_l == 0:
            raise ValueError('inputsize input is zero')
        Input_bytes_second_res = inputsize_l / downloadtime_l
        return Input_bytes_second_res

    def Input_bytes_calc(self, downloadtime_l, inputsize_l, Nr_Evts):
        "input size [bytes]"
        if inputsize_l == 0:
            raise ValueError('inputsize input is zero')
        Input_bytes_res = inputsize_l * self.Total_Events / Nr_Evts
        return Input_bytes_res
```

```
    def DownloadTime_calc(self, downloadtime_l, inputsize_l, Nr_Evts):
        "downloadtime [second]"
        if downloadtime_l == 0:
            raise ValueError('downloadtime input is zero')
        downloadtime = downloadtime_l / Nr_Evts * self.Total_Events
        return downloadtime

    def CPUTime_calc(self, CPUTime_l, Nr_Evts):
        "CPU Time [seconds]"
        CPUTime = CPUTime_l * self.Total_Events
        return CPUTime
```

uncertainty_estimation.py

```
from math import sqrt

class estimate_uncertainty:
    "uncertainty estimations"
    def __init__(self, Nr_Evts):
        self.Nr_Evts = Nr_Evts

    def Time_calc_unc(self):
        "calculate the wall time uncertainty. error combination: idle
        ↪  and cpu time dependent, io time independent"
        time_Uncertainty =
        ↪  sqrt((self.CPU_Idle_Time_stdev/self.Nr_Cores)**2 +
        ↪  (self.CPU_Time_stdev/self.Nr_Cores)**2 +
        ↪  (self.IO_Time_stdev/self.Nr_Cores)**2)
        return time_Uncertainty

    def num_workflows_unc(self):
        "calculate the sum workflows"
        num_workflow_Uncertainty = (self.infrastructure_duration *
        ↪  self.nr_machines * self.Nr_Cores /
        ↪  (self.total_processing_time)**2 *
                                    self.Time_calc_unc())
        return num_workflow_Uncertainty

    def total_events_unc(self):
        "calculate the total_events uncertainty"
        total_events = self.Nr_Evts * self.overcommit_factor *
        ↪  self.num_workflows_unc()
        return total_events
```

```
def sum_walltime_unc(self):
    "calculate the sum wall time uncertainty"
    sum_walltime = 0
    return sum_walltime

def ETC_calc_unc(self):
    "calculate the ETC uncertainty"
    ETC_Uncertainty = (self.Time_calc_unc() * self.Nr_Cores /
    ↪  (self.total_processing_time)**2 /
                       self.total_cost_machines * self.Nr_Evts *
                       ↪  self.overcommit_factor)
    return ETC_Uncertainty

def EventCost_calc_unc(self):
    "calculate the EC uncertainty"
    EC_Uncertainty = self.total_events_unc() /
    ↪  self.total_cost_machines
    return EC_Uncertainty

def EventTime_calc_unc(self):
    "calculate the ET uncertainty"
    ET_Uncertainty = (self.total_events_unc() / self.sum_walltimes)
    return ET_Uncertainty

def Time_all_calc_unc(self):
    "uncertainty on duration of excercise"
    time_all_Uncertainty = 0
    return time_all_Uncertainty

def Bandwidth_calc_unc(self):
    "calculate the bandwidth uncertainty"
    Bandwidth_Uncertainty = (self.input_size *
    ↪  self.total_events_unc() / self.sum_walltimes / 1000000000 *
    ↪  8)
    return Bandwidth_Uncertainty

def CostEvent_calc_unc(self):
    "calculate the cost/event uncertainty"
    cost_event_Uncertainty = self.total_cost_machines /
    ↪  (self.Total_Events)**2 * self.total_events_unc()
    return cost_event_Uncertainty
```

```
def Events_calc_unc(self):
    "calculate the #event uncertainty"
    event_Uncertainty = self.total_events_unc()
    return event_Uncertainty

def Input_bytes_second_calc_unc(self):
    "calculate the input speed uncertainty"
    input_speed_Uncertainty = 0
    return input_speed_Uncertainty

def Input_bytes_calc_unc(self):
    "calculate the input size uncertainty"
    input_size_Uncertainty = inputsize_l * self.total_events_unc()
    ↪  / Nr_Evts
    return input_size_Uncertainty

def DownloadTime_calc_unc(self):
    "calculate the download duration uncertainty"
    download_time_Uncertainty = downloadtime_l / Nr_Evts *
    ↪  self.total_events_unc()
    return download_time_Uncertainty

def CPUTime_calc_unc(self):
    "calculate the CPU time uncertainty"
    CPU_time_Uncertainty = CPUTime_l * self.total_events_unc()
    return CPU_time_Uncertainty
```

A.4 Overcommitting

Processes	Wall [s]	Wall STDEV [%]	CPU [s]	Idle [s]	I/O Wait [s]
4	11306	0.46	37388	52554	152
5	9836	0.12	38259	40029	156
6	8898	0.24	38688	32010	194
7	8665	1.16	40309	28214	416
8	8427	0.28	42044	24016	1140
9	8839	0.12	43940	24838	1644
10	9328	0.17	45817	25962	2470

Table A1: Job duration with increasing number of parallel processes.

A.4.1 Model input parameters

Budget = 1000
CPU power = 10
Bandwidth = 1250000000
Ram-to-CPU-ratio = 0.358

A.5 Additional source code and scripts

```c
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/mman.h>

int main(int argc, char argv) {
  size_t s = (unsigned long long int) atol(argv[1]);
  char c=(char)malloc(s);
  mlock(c, s);
  memset(c,0,s);
  sleep(86400);
  return 0;
}
```

Listing 1: Code to allocate and lock the amount of RAM that is specified upon execution of the program for 86400 seconds.

Listing 2: "lscpu" result on the 4 core VM at CERN.

```
[root@datacloud02 ~]# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             4
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 44
Model name:            Intel(R) Xeon(R) CPU           L5640  @
  2.27GHz
Stepping:              2
CPU MHz:               2266.746
BogoMIPS:              4533.49
```

```
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              12288K
NUMA node0 CPU(s):     0-3
```

Listing 3: Monitoring with the 'sar' commands of the sysstat package.

```
sar -P ALL 5 20000 > sar_P_ALL_CPU.txt &
sar -u ALL 5 20000 > sar_u.txt &
sar -n DEV 5 20000 > sar_n_DEV_network.txt &
sar -b 5 20000 > sar_b_IO.txt &
sar -r 5 20000 > sar_r_memory.txt &
sar -S 5 20000 > sar_S_swap.txt &
sar -B 5 20000 > sar_B_paging_statistics.txt &
sar -W 5 20000 > sar_W_page_swap_statisitcs.txt &
sar -p -d 5 20000 > sar_d_individual_block_device.txt &
sar -w 5 20000 > sar_w_context_switch.txt &
sar -q 5 20000 > sar_q_load_average.txt &
```

Listing 4: Monitoring with the /proc pseudo-file system.

```
cat /proc/stat > proc_stat_beginning.txt
cat /proc/uptime > proc_uptime_start.txt
cat /proc/uptime > proc_uptime.txt
while kill -0 $pid2 2> /dev/null
do
    cat /proc/$pid2/stat > proc_pid_stat.txt
    cat /proc/$pid2/io > proc_pid_io.txt
    cat /proc/$pid2/status > proc_pid_status.txt
    cat /proc/stat > proc_stat_end.txt
    cat /proc/uptime > proc_uptime_end.txt
    function getcpid() {
    cpids=`pgrep -P $1|xargs`
    for cpid in $cpids;
    do
        cat /proc/$cpid/stat > proc_pid_stat_$cpid.txt
        cat /proc/$cpid/io > proc_pid_io_$cpid.txt
        cat /proc/$cpid/status > proc_pid_status_$cpid.txt
        getcpid $cpid
    done
    }
    getcpid $pid2
```

```
22          sleep 5s
23      done
```

Listing 5: Monitoring with the /sys pseudo-file system.

```
1       echo `date` >> disk_stat.txt
2       echo "third value 512 is bytes read, 7th value
          512 bytes written" >> disk_stat.txt
3       echo "vda3, /swap" >> disk_stat.txt
4       tail /sys/block/vda/vda3/stat >> disk_stat.txt
5       echo "vda, alles" >> disk_stat.txt
6       tail /sys/block/vda/stat >> disk_stat.txt
7       echo "vda4, /test " >> disk_stat.txt
8       tail /sys/block/vda/vda4/stat >> disk_stat.txt
9       echo "vdb /storage" >> disk_stat.txt
10      tail /sys/block/vdb/stat >> disk_stat.txt
```

A.6 Different workflows

A.6.1 Event Generation

Athena version: asetup –cmtconfig=x86_64-slc6-gcc47-opt AtlasProduction,19.2.3.6,notest

The command line input was:

```
Generate_tf.py --AMITag=e3735 --ecmEnergy=13000
↪ --evgenJobOpts=/test/evgen/MC15JobOpts-00-00-33_v2.tar.gz
↪ --jobConfig=/test/evgen/MC15.424100.Pythia8B_A14_CTEQ6L1_Jpsimu
↪ 4mu4.py --maxEvents=1000
↪ --outputEVNTFile=EVNT.11321089._026174.pool.root.1
↪ --randomSeed=70575..70600 --runNumber=424100 --skipEvents=0 &>
↪ output.txt &
```

The file ‘MC15.424100.Pythia8B_A14_CTEQ6L1_Jpsimu4mu4.py’ contains:

```
#--------------------------------------------------------------
# JO  fragment for pp->J/psi(mu4mu4)X, Pythia 8B
#--------------------------------------------------------------

evgenConfig.description = "Inclusive pp->J/psi(mu4mu4) production with
↪  Photos"
evgenConfig.keywords = ["charmonium","2muon","inclusive"]
evgenConfig.minevents = 500

include('MC15JobOptions/nonStandard/Pythia8B_A14_CTEQ6L1_Common.py')
include('MC15JobOptions/nonStandard/Pythia8B_Photospp.py')
```

```
include("MC15JobOptions/Pythia8B_Charmonium_Common.py")

genSeq.Pythia8B.Commands += ['PhaseSpace:pTHatMin = 4.']
genSeq.Pythia8B.Commands += ['443:onMode = off']
genSeq.Pythia8B.Commands += ['443:2:onMode = on']
genSeq.Pythia8B.SignalPDGCodes = [443,-13,13]

genSeq.Pythia8B.TriggerPDGCode = 13
genSeq.Pythia8B.TriggerStatePtCut = [4.0]
genSeq.Pythia8B.TriggerStateEtaCut = 2.7
genSeq.Pythia8B.MinimumCountPerCut = [2]
```

A.6.2 Monte-Carlo simulation

Input files are the outputs from the event generation in Subsection A.6.1. Athena version: asetup –cmtconfig=x86_64-slc6-gcc49-opt AtlasOffline,21.0.15,notest,here

The command line input was:

```
Sim_tf.py --inputEVNTFile=/test/evgen/"counter"_1000evts/EVNT.11321
↪ 089._026174.pool.root.1 --maxEvents=100 --postInclude
↪ default:RecJobTransforms/UseFrontier.py
↪ --athenaMPMergeTargetSize=ALL:0.0 --preExec 'EVNTtoHITS:simFlag
↪ s.SimBarcodeOffset.set_Value_and_Lock(200000)'
↪ 'EVNTtoHITS:simFlags.TRTRangeCut=30.0;
↪ simFlags.TightMuonStepping=True' --preInclude
↪ EVNTtoHITS:SimulationJobOptions/preInclude.BeamPipeKill.py,Simu
↪ lationJobOptions/preInclude.FrozenShowersFCalOnly.py
↪ --skipEvents=0 --firstEvent=0
↪ --outputHITSFile=HITS.11363361._041300.pool.root.1
↪ --physicsList=FTFP_BERT_ATL_VALIDATION --randomSeed=13056
↪ --DBRelease=all:current --conditionsTag
↪ default:OFLCOND-MC16-SDR-14
↪ --geometryVersion=default:ATLAS-R2-2016-01-00-01_VALIDATION
↪ --runNumber=361100 --AMITag=s3126 --DataRunNumber=284500
↪ --simulator=FullG4 --truthStrategy=MC15aPlus &> output.txt &
```

A.6.3 Reconstruction 1

Uses the same input file for all iterations. Athena version: asetup –cmtconfig=x86_64-slc6-gcc49-opt AtlasProduction,20.7.7.6

The command line input was:

```
Reco_tf.py '--inputBSFile=/test/reco/data16_13TeV.00304008.physics_
↪ Main.daq.RAW._lb0167._SFO-4._0002.data'
↪ '--maxEvents=default:-1' '--postExec' 'ESDtoDPD:from
↪ AthenaCommon.AppMgr import ServiceMgr;import
↪ MuonRPC_Cabling.MuonRPC_CablingConfig;ServiceMgr.MuonRPC_Cablin
↪ gSvc.RPCMapfromCool=False;ServiceMgr.MuonRPC_CablingSvc.CorrFil
↪ eName="LVL1confAtlasRUN2_ver016.corr";ServiceMgr.MuonRPC_Cablin
↪ gSvc.ConfFileName="LVL1confAtlasRUN2_ver016.data";' '--preExec'
↪ 'all:DQMonFlags.enableLumiAccess=False;'
↪ '--autoConfiguration=everything' '--beamType=collisions'
↪ '--conditionsTag' 'default:CONDBR2-BLKPA-2016-16'
↪ '--geometryVersion=default:ATLAS-R2-2015-04-00-00'
↪ '--runNumber=304008' '--AMITag=r8519'
↪ '--outputAODFile=AOD.09455490._003112.pool.root.1'
↪ '--outputHISTFile=HIST.09455490._003112.pool.root.1'
↪ '--jobNumber=3106' '--ignoreErrors=True'
↪ '--ignorePatterns=ToolSvc.InDetSCTRodDecoder.+ERROR.+Unknown.+o
↪ fflineId.+for.+OnlineId'  &> output.txt
↪ &
```

A.6.4 Reconstruction 2

Same input file for all iterations.

Athena version: asetup –cmtconfig=x86_64-slc6-gcc62-opt Athena,21.0.30

The command line input was:

```
Reco_tf.py --inputBSFile=/test/reco/17data_1/data17_13TeV.00329964.
↪ physics_Main.daq.RAW._lb0366._SFO-8._0003.data --maxEvents -1
↪ --postExec 'e2d:from AthenaCommon.AppMgr import ServiceMgr;
import MuonRPC_Cabling.MuonRPC_CablingConfig;
ServiceMgr.MuonRPC_CablingSvc.RPCMapfromCool=False;
ServiceMgr.MuonRPC_CablingSvc.CorrFileName="LVL1confAtlasRUN2_ver10
↪ 4.corr";
ServiceMgr.MuonRPC_CablingSvc.ConfFileName="LVL1confAtlasRUN2_ver10
↪ 4.data";
' 'e2a:if "TileJetMonTool/TileJetMonTool" in ToolSvc.getSequence():
↪ ToolSvc.TileJetMonTool.do_1dim_histos=True;
' --preExec 'all:from InDetRecExample.InDetJobProperties import
↪ InDetFlags;
 InDetFlags.useDynamicAlignFolders.set_Value_and_Lock(True);
from InDetPrepRawDataToxAOD.SCTxAODJobProperties import
↪ SCTxAODFlags;
SCTxAODFlags.Prescale.set_Value_and_Lock(50);
TriggerFlags.AODEDMSet="AODFULL";
```

```
from TrigHLTMonitoring.HLTMonFlags import HLTMonFlags;
HLTMonFlags.doGeneral=False;
HLTMonFlags.doJet=False' --autoConfiguration everything
 ↪  --conditionsTag all:CONDBR2-BLKPA-2017-10 --geometryVersion
 ↪  all:ATLAS-R2-2016-01-00-01 --runNumber 329964 --AMITag f844 --o
 ↪  utputDESDM_CALJETFile=DESDM_CALJET.11776717._004044.pool.root.1
 ↪  --outputDESDM_EXOTHIPFile=DESDM_EXOTHIP.11776717._004044.pool.r
 ↪  oot.1
 ↪  --outputDESDM_MCPFile=DESDM_MCP.11776717._004044.pool.root.1
 ↪  --outputDESDM_PHOJETFile=DESDM_PHOJET.11776717._004044.pool.roo
 ↪  t.1
 ↪  --outputDESDM_SGLELFile=DESDM_SGLEL.11776717._004044.pool.root.
 ↪  1
 ↪  --outputDESDM_TILEMUFile=DESDM_TILEMU.11776717._004044.pool.roo
 ↪  t.1 --outputDRAW_EGZFile=DRAW_EGZ.11776717._004044.pool.root.1
 ↪  --outputDRAW_RPVLLFile=DRAW_RPVLL.11776717._004044.pool.root.1
 ↪  --outputDRAW_TAUMUHFile=DRAW_TAUMUH.11776717._004044.pool.root.
 ↪  1
 ↪  --outputDRAW_ZMUMUFile=DRAW_ZMUMU.11776717._004044.pool.root.1
 ↪  --outputAODFile=AOD.11776717._004044.pool.root.1
 ↪  --outputHISTFile=HIST.11776717._004044.pool.root.1 --jobNumber
 ↪  3941 --ignoreErrors false  &> output.txt &
```

A.6.5 Reconstruction 3

Athena version: asetup –cmtconfig=x86_64-slc6-gcc62-opt Athena,21.0.30 Different input file for each job inside a VM (same input file across VMs), e.g.: data17_13TeV.00325030.physics_Main.daq.RAW._lb0685._SFO-7._0001.data, data17_13TeV.00325030.physics_ Main.daq.RAW._lb0685._SFO-5._0001.data, ...

The command line input was:

```
Reco_tf.py --inputBSFile=/test/reco/counter --maxEvents -1
 ↪  --postExec 'e2d:from AthenaCommon.AppMgr import ServiceMgr;
import MuonRPC_Cabling.MuonRPC_CablingConfig;
ServiceMgr.MuonRPC_CablingSvc.RPCMapfromCool=False;
ServiceMgr.MuonRPC_CablingSvc.CorrFileName="LVL1confAtlasRUN2_ver10
 ↪  4.corr";
ServiceMgr.MuonRPC_CablingSvc.ConfFileName="LVL1confAtlasRUN2_ver10
 ↪  4.data";
' 'e2a:if "TileJetMonTool/TileJetMonTool" in ToolSvc.getSequence():
 ↪  ToolSvc.TileJetMonTool.do_1dim_histos=True;
' --preExec 'all:from InDetRecExample.InDetJobProperties import
 ↪  InDetFlags;
 InDetFlags.useDynamicAlignFolders.set_Value_and_Lock(True);
```

```
from InDetPrepRawDataToxAOD.SCTxAODJobProperties import
↪  SCTxAODFlags;
SCTxAODFlags.Prescale.set_Value_and_Lock(50);
TriggerFlags.AODEDMSet="AODFULL";
from TrigHLTMonitoring.HLTMonFlags import HLTMonFlags;
HLTMonFlags.doGeneral=False;
HLTMonFlags.doJet=False' --autoConfiguration everything
↪  --conditionsTag all:CONDBR2-BLKPA-2017-10 --geometryVersion
↪  all:ATLAS-R2-2016-01-00-01 --runNumber 329964 --AMITag f844 --o
↪  utputDESDM_CALJETFile=DESDM_CALJET.11776717._004044.pool.root.1
↪  --outputDESDM_EXOTHIPFile=DESDM_EXOTHIP.11776717._004044.pool.r
↪  oot.1
↪  --outputDESDM_MCPFile=DESDM_MCP.11776717._004044.pool.root.1
↪  --outputDESDM_PHOJETFile=DESDM_PHOJET.11776717._004044.pool.roo
↪  t.1
↪  --outputDESDM_SGLELFile=DESDM_SGLEL.11776717._004044.pool.root.
↪  1
↪  --outputDESDM_TILEMUFile=DESDM_TILEMU.11776717._004044.pool.roo
↪  t.1 --outputDRAW_EGZFile=DRAW_EGZ.11776717._004044.pool.root.1
↪  --outputDRAW_RPVLLFile=DRAW_RPVLL.11776717._004044.pool.root.1
↪  --outputDRAW_TAUMUHFile=DRAW_TAUMUH.11776717._004044.pool.root.
↪  1
↪  --outputDRAW_ZMUMUFile=DRAW_ZMUMU.11776717._004044.pool.root.1
↪  --outputAODFile=AOD.11776717._004044.pool.root.1
↪  --outputHISTFile=HIST.11776717._004044.pool.root.1 --jobNumber
↪  3941 --ignoreErrors false  &> output.txt &
```

A.6.6 Reconstruction 4

Athena version: asetup –cmtconfig=x86_64-slc6-gcc62-opt Athena,21.0.30 Different input file for each job inside a VM (same input file across VMs) e.g.: data17_13TeV.00325030.physics_Main.daq.RAW._lb0684._SFO-3._0001.data, data17_13TeV.00325030.physics_Main.daq.RAW._lb0660._SFO-8._0001.data, ...

The command line input was:

```
Reco_tf.py --inputBSFile=/test/reco/counter --maxEvents -1
↪  --postExec 'e2d:from AthenaCommon.AppMgr import ServiceMgr;
import MuonRPC_Cabling.MuonRPC_CablingConfig;
ServiceMgr.MuonRPC_CablingSvc.RPCMapfromCool=False;
ServiceMgr.MuonRPC_CablingSvc.CorrFileName="LVL1confAtlasRUN2_ver10
↪  4.corr";
ServiceMgr.MuonRPC_CablingSvc.ConfFileName="LVL1confAtlasRUN2_ver10
↪  4.data";
```

```
' 'e2a:if "TileJetMonTool/TileJetMonTool" in ToolSvc.getSequence():
↪  ToolSvc.TileJetMonTool.do_1dim_histos=True;
' --preExec 'all:from InDetRecExample.InDetJobProperties import
↪  InDetFlags;
 InDetFlags.useDynamicAlignFolders.set_Value_and_Lock(True);
from InDetPrepRawDataToxAOD.SCTxAODJobProperties import
↪  SCTxAODFlags;
SCTxAODFlags.Prescale.set_Value_and_Lock(50);
TriggerFlags.AODEDMSet="AODFULL";
from TrigHLTMonitoring.HLTMonFlags import HLTMonFlags;
HLTMonFlags.doGeneral=False;
HLTMonFlags.doJet=False' --autoConfiguration everything
↪  --conditionsTag all:CONDBR2-BLKPA-2017-10 --geometryVersion
↪  all:ATLAS-R2-2016-01-00-01 --runNumber 329964 --AMITag f844 --o
↪  utputDESDM_CALJETFile=DESDM_CALJET.11776717._004044.pool.root.1
↪  --outputDESDM_EXOTHIPFile=DESDM_EXOTHIP.11776717._004044.pool.r
↪  oot.1
↪  --outputDESDM_MCPFile=DESDM_MCP.11776717._004044.pool.root.1
↪  --outputDESDM_PHOJETFile=DESDM_PHOJET.11776717._004044.pool.roo
↪  t.1
↪  --outputDESDM_SGLELFile=DESDM_SGLEL.11776717._004044.pool.root.
↪  1
↪  --outputDESDM_TILEMUFile=DESDM_TILEMU.11776717._004044.pool.roo
↪  t.1 --outputDRAW_EGZFile=DRAW_EGZ.11776717._004044.pool.root.1
↪  --outputDRAW_RPVLLFile=DRAW_RPVLL.11776717._004044.pool.root.1
↪  --outputDRAW_TAUMUHFile=DRAW_TAUMUH.11776717._004044.pool.root.
↪  1
↪  --outputDRAW_ZMUMUFile=DRAW_ZMUMU.11776717._004044.pool.root.1
↪  --outputAODFile=AOD.11776717._004044.pool.root.1
↪  --outputHISTFile=HIST.11776717._004044.pool.root.1 --jobNumber
↪  3941 --ignoreErrors false  &> output.txt &
```

A.6.7 Reconstruction 5

Different input file for each job inside a VM (same input file across VMs), e.g.: data17_13TeV.00325030.physics_Main.daq.RAW._lb0684._SFO-3._0001.data, data17_13TeV.00325030.physics_Main.daq.RAW._lb0660._SFO-8._0001.data, ...

Athena version: asetup –cmtconfig=x86_64-slc6-gcc62-opt Athena,21.0.37,notest

The command line input was:

```
Reco_tf.py --inputBSFile=/test/reco/counter --maxEvents=default:-1
↪  --postExec 'all:from AthenaCommon.AppMgr import ServiceMgr as
↪  svcMgr;
 svcMgr.AthenaPoolCnvSvc.MaxFileSizes=["15000000000"];
```

```
from IOVDbSvc.CondDB import conddb;
 conddb.addOverride("/MUONALIGN/CSC/ILINES",
  ↪  "MuonAlignCscIlines-UPD1-03");
' 'ESDtoDPD:from AthenaCommon.AppMgr import ServiceMgr;
import MuonRPC_Cabling.MuonRPC_CablingConfig;
ServiceMgr.MuonRPC_CablingSvc.RPCMapfromCool=False;
ServiceMgr.MuonRPC_CablingSvc.CorrFileName="LVL1confAtlasRUN2_ver10
↪  4.corr";
ServiceMgr.MuonRPC_CablingSvc.ConfFileName="LVL1confAtlasRUN2_ver10
↪  4.data";
' --preExec 'all:from InDetRecExample.InDetJobProperties import
↪  InDetFlags;
InDetFlags.useDynamicAlignFolders.set_Value_and_Lock(True);
 from PrimaryDPDMaker.PrimaryDESDMFlags_PerfMS import
  ↪  PrimaryDESDMFlags_PerfMSStream;
 jobproperties.PrimaryDESDMFlags_PerfMSStream.doAlignmentFormat.set
  ↪  _Value_and_Lock(True);
' --skipEvents=0 --conditionsTag all:CONDBR2-BLKPA-2017-12
↪  --geometryVersion=all:ATLAS-R2-2016-01-00-01 --runNumber=325030
↪  --AMITag=r9962 --outputDESDM_ALLCELLSFile=DESDM_MCP.12214515._0
↪  25888.pool.root.1 --jobNumber=20791 &> output.txt
↪  &
```

A.6.8 Reconstruction 6

Software version and input data from the beginning of 2015. Different input file for each job inside a VM (same input file across VMs), e.g. data15_13TeV.00270588.physics_Main.daq.RAW._lb0233._SFO-2._0001.data, data15_13TeV.00270588.physics_Main.daq.RAW._lb0238._SFO-1._0001.data, ... Athena version: asetup –cmtconfig=x86_64-slc6-gcc48-opt 20.1.5.5,AtlasProduction,here,notest

The command line input was:

```
Reco_tf.py --inputBSFile=/test/reco_beg_run/data15_13TeV.00270588.physi
↪  cs_Main.daq.RAW._lb0233._SFO-2._0001.data --maxEvents=default:-1
↪  --postExec 'ESDtoDPD:from AthenaCommon.AppMgr import ServiceMgr;
import MuonRPC_Cabling.MuonRPC_CablingConfig;
ServiceMgr.MuonRPC_CablingSvc.RPCMapfromCool=False;
ServiceMgr.MuonRPC_CablingSvc.CorrFileName="LVL1confAtlasRUN2_ver016.co
↪  rr";
ServiceMgr.MuonRPC_CablingSvc.ConfFileName="LVL1confAtlasRUN2_ver016.da
↪  ta";
' 'RAWtoESD:from AthenaCommon.AppMgr import ServiceMgr as svcMgr;
svcMgr.AthenaPoolCnvSvc.MaxFileSizes=["15000000000"];
```

```
' 'ESDtoAOD:CILMergeAOD.removeItem("xAOD::CaloClusterAuxContainer#CaloC
↪  alTopoClustersAux.LATERAL.LONGITUDINAL.SECOND_R.SECOND_LAMBDA.CENTE
↪  R_MAG.CENTER_LAMBDA.FIRST_ENG_DENS.ENG_FRAC_MAX.ISOLATION.ENG_BAD_C
↪  ELLS.N_BAD_CELLS.BADLARQ_FRAC.ENG_BAD_HV_CELLS.N_BAD_HV_CELLS.ENG_P
↪  OS.SIGNIFICANCE.CELL_SIGNIFICANCE.CELL_SIG_SAMPLING.AVG_LAR_Q.AVG_T
↪  ILE_Q.EM_PROBABILITY.PTD.BadChannelList");
CILMergeAOD.add("xAOD::CaloClusterAuxContainer#CaloCalTopoClustersAux.N
↪  _BAD_CELLS.ENG_BAD_CELLS.BADLARQ_FRAC.AVG_TILE_Q.AVG_LAR_Q.CENTER_M
↪  AG.ENG_POS.CENTER_LAMBDA.SECOND_LAMBDA.SECOND_R.ISOLATION.EM_PROBAB
↪  ILITY");
StreamAOD.ItemList=CILMergeAOD();
' --preExec
↪  'DQMonFlags.enableLumiAccess=False;DQMonFlags.doCTPMon=False;from
↪  MuonRecExample.MuonRecFlags import muonRecFlags;muonRecFlags.useLoo
↪  seErrorTuning.set_Value_and_Lock(True);
DQMonFlags.enableLumiAccess=False;
from InDetRecExample.InDetJobProperties import InDetFlags;
from BTagging.BTaggingFlags import BTaggingFlags;
BTaggingFlags.btaggingAODList=["xAOD::BTaggingContainer#BTagging_AntiKt
↪  4EMTopo","xAOD::BTaggingAuxContainer#BTagging_AntiKt4EMTopoAux.","x
↪  AOD::BTagVertexContainer#BTagging_AntiKt4EMTopoJFVtx","xAOD::BTagVe
↪  rtexAuxContainer#BTagging_AntiKt4EMTopoJFVtxAux.","xAOD::VertexCont
↪  ainer#BTagging_AntiKt4EMTopoSecVtx","xAOD::VertexAuxContainer#BTagg
↪  ing_AntiKt4EMTopoSecVtxAux.-vxTrackAtVertex"];
' 'RAWtoESD:from InDetRecExample.InDetJobProperties import InDetFlags;
 InDetFlags.cutLevel.set_Value_and_Lock(14);
 from JetRec import JetRecUtils;
f=lambda s:["xAOD::JetContainer#AntiKt4\%sJets"\%(s,),"xAOD::JetAuxCont
↪  ainer#AntiKt4\%sJetsAux."\%(s,),"xAOD::EventShape#Kt4\%sEventShape"
↪  \%(s,),"xAOD::EventShapeAuxInfo#Kt4\%sEventShapeAux."\%(s,),"xAOD::
↪  EventShape#Kt4\%sOriginEventShape"\%(s,),"xAOD::EventShapeAuxInfo#K
↪  t4\%sOriginEventShapeAux."\%(s,)];
```

```
 JetRecUtils.retrieveAODList = lambda :
  ↪  f("EMPFlow")+f("LCTopo")+f("EMTopo")+["xAOD::EventShape#NeutralPar
  ↪  ticleFlowIsoCentralEventShape","xAOD::EventShapeAuxInfo#NeutralPar
  ↪  ticleFlowIsoCentralEventShapeAux.","xAOD::EventShape#NeutralPartic
  ↪  leFlowIsoForwardEventShape","xAOD::EventShapeAuxInfo#NeutralPartic
  ↪  leFlowIsoForwardEventShapeAux.","xAOD::EventShape#ParticleFlowIsoC
  ↪  entralEventShape","xAOD::EventShapeAuxInfo#ParticleFlowIsoCentralE
  ↪  ventShapeAux.","xAOD::EventShape#ParticleFlowIsoForwardEventShape"
  ↪  ,"xAOD::EventShapeAuxInfo#ParticleFlowIsoForwardEventShapeAux.","x
  ↪  AOD::EventShape#TopoClusterIsoCentralEventShape","xAOD::EventShape
  ↪  AuxInfo#TopoClusterIsoCentralEventShapeAux.","xAOD::EventShape#Top
  ↪  oClusterIsoForwardEventShape","xAOD::EventShapeAuxInfo#TopoCluster
  ↪  IsoForwardEventShapeAux.","xAOD::CaloClusterContainer#EMOriginTopo
  ↪  Clusters","xAOD::ShallowAuxContainer#EMOriginTopoClustersAux.","xA
  ↪  OD::CaloClusterContainer#LCOriginTopoClusters","xAOD::ShallowAuxCo
  ↪  ntainer#LCOriginTopoClustersAux."];
from eflowRec.eflowRecFlags import jobproperties;
 jobproperties.eflowRecFlags.useAODReductionClusterMomentList.set_Value
  ↪  _and_Lock(True);
 from TriggerJobOpts.TriggerFlags import TriggerFlags;
TriggerFlags.AODEDMSet.set_Value_and_Lock("AODFULL");
from LArConditionsCommon.LArCondFlags import larCondFlags;
larCondFlags.OFCShapeFolder.set_Value_and_Lock("4samples1phase");
' 'ESDtoAOD:from ParticleBuilderOptions.AODFlags import AODFlags;
AODFlags.ThinNegativeEnergyCaloClusters.set_Value_and_Lock(True);
AODFlags.ThinNegativeEnergyNeutralPFOs.set_Value_and_Lock(True);
 from JetRec import JetRecUtils;
 aodlist = JetRecUtils.retrieveAODList();
 JetRecUtils.retrieveAODList = lambda : [item for item in aodlist if
  ↪  not "OriginTopoClusters" in item];
' --autoConfiguration=everything --beamType=collisions --conditionsTag
 ↪  default:CONDBR2-BLKPA-2015-05
 ↪  --geometryVersion=ATLAS-R2-2015-03-01-00 --runNumber=270588
 ↪  --AMITag=f594
 ↪  --outputDESDM_EGAMMAFile=DESDM_EGAMMA.11258128._000821.pool.root.1
 ↪  --outputDRAW_EGZFile=DRAW_EGZ.11258128._000821.pool.root.1
 ↪  --outputDRAW_EMUFile=DRAW_EMU.11258128._000821.pool.root.1
 ↪  --outputDRAW_TAUMUHFile=DRAW_TAUMUH.11258128._000821.pool.root.1
 ↪  --outputDRAW_ZMUMUFile=DRAW_ZMUMU.11258128._000821.pool.root.1
 ↪  --outputAODFile=AOD.11258128._000821.pool.root.1
 ↪  --outputHISTFile=HIST.11258128._000821.pool.root.1 --jobNumber=821
 ↪  &> output.txt &
```

A.6.9 Reconstruction 7

Different input file for each job inside a VM and also different input files across VMs

Athena version: asetup –cmtconfig=x86_64-slc6-gcc62-opt Athena,21.0.39,notest

The command line input was:

```
Reco_tf.py --inputBSFile=/test/reco_diffdiff/*.data --postExec
↪  'all:from AthenaCommon.AppMgr import ServiceMgr as svcMgr;
 svcMgr.AthenaPoolCnvSvc.MaxFileSizes=["15000000000"];
' --preExec 'all:from InDetRecExample.InDetJobProperties import
↪  InDetFlags;
InDetFlags.useDynamicAlignFolders.set_Value_and_Lock(True);
' 'ESDtoDPD:from AthenaCommon.AppMgr import ServiceMgr;
import MuonRPC_Cabling.MuonRPC_CablingConfig;
ServiceMgr.MuonRPC_CablingSvc.RPCMapfromCool=False;
ServiceMgr.MuonRPC_CablingSvc.CorrFileName="LVL1confAtlasRUN2_ver10
↪  4.corr";
ServiceMgr.MuonRPC_CablingSvc.ConfFileName="LVL1confAtlasRUN2_ver10
↪  4.data";
```

```
' --conditionsTag all:CONDBR2-BLKPA-2017-13
↪  --geometryVersion=all:ATLAS-R2-2016-01-00-01 --runNumber=331085
↪  --AMITag=r10053 --outputDAOD_IDTIDEFile=DAOD_IDTIDE.12444252._0
↪  16806.pool.root.1
↪  --outputDESDM_CALJETFile=DESDM_CALJET.12444252._016806.pool.roo
↪  t.1
↪  --outputDESDM_EGAMMAFile=DESDM_EGAMMA.12444252._016806.pool.roo
↪  t.1
↪  --outputDESDM_EXOTHIPFile=DESDM_EXOTHIP.12444252._016806.pool.r
↪  oot.1
↪  --outputDESDM_MCPFile=DESDM_MCP.12444252._016806.pool.root.1
↪  --outputDESDM_PHOJETFile=DESDM_PHOJET.12444252._016806.pool.roo
↪  t.1
↪  --outputDESDM_SGLELFile=DESDM_SGLEL.12444252._016806.pool.root.
↪  1
↪  --outputDESDM_SLTTMUFile=DESDM_SLTTMU.12444252._016806.pool.roo
↪  t.1 --outputDRAW_EGZFile=DRAW_EGZ.12444252._016806.pool.root.1
↪  --outputDRAW_EMUFile=DRAW_EMU.12444252._016806.pool.root.1
↪  --outputDRAW_RPVLLFile=DRAW_RPVLL.12444252._016806.pool.root.1
↪  --outputDRAW_TAUMUHFile=DRAW_TAUMUH.12444252._016806.pool.root.
↪  1
↪  --outputDRAW_TOPSLMUFile=DRAW_TOPSLMU.12444252._016806.pool.roo
↪  t.1
↪  --outputDRAW_ZMUMUFile=DRAW_ZMUMU.12444252._016806.pool.root.1
↪  --outputAODFile=AOD.12444252._016806.pool.root.1
↪  --outputHISTFile=HIST.12444252._016806.pool.root.1
↪  --jobNumber=10016 &> output.txt &
```

A.6.10 Digitisation and reconstruction 1

Athena version: asetup –cmtconfig=x86_64-slc6-gcc62-opt AtlasOffline,21.0.20

The command line input was:

```
Reco_tf.py --inputHITSFile=/test/digireco/HITS.12118682._003334.poo
↪ l.root.1,/test/digireco/HITS.12118682._003335.pool.root.1
↪ --maxEvents=200 --postExec
↪ "all:CfgMgr.MessageSvc().setError+=[\"HepMcParticleLink\"]"
↪ "RDOtoRDOTrigger:conddb.addOverride(\"/CALO/Ofl/Noise/PileUpNoi
↪ seLumi\",\"CALOOflNoisePileUpNoiseLumi-mc15-mu30-dt25ns\")"
↪ --postInclude "default:PyJobTransforms/UseFrontier.py"
↪ --preExec "all:rec.Commissioning.set_Value_and_Lock(True);from
↪ AthenaCommon.BeamFlags import jobproperties;jobproperties.Beam.
↪ numberOfCollisions.set_Value_and_Lock(20.0);from
↪ LArROD.LArRODFlags import larRODFlags;larRODFlags.NumberOfColli
↪ sions.set_Value_and_Lock(20);larRODFlags.nSamples.set_Value_and
↪ _Lock(4);larRODFlags.doOFCPileupOptimization.set_Value_and_Lock
↪ (True);larRODFlags.firstSample.set_Value_and_Lock(0);larRODFlag
↪ s.useHighestGainAutoCorr.set_Value_and_Lock(True)" "all:from
↪ TriggerJobOpts.TriggerFlags import TriggerFlags as
↪ TF;TF.run2Config='2016'" --preInclude
↪ "HITtoRDO:Digitization/ForceUseOfPileUpTools.py,SimulationJobOp
↪ tions/preInclude.PileUpBunchTrainsMC15_2015_25ns_Config1.py,Run
↪ DependentSimData/configLumi_run284500_mc16a.py" --skipEvents=0
↪ --autoConfiguration=everything --conditionsTag
↪ "default:OFLCOND-MC16-SDR-16"
↪ --geometryVersion="default:ATLAS-R2-2016-01-00-01"
↪ --runNumber=363868 --digiSeedOffset1=1664
↪ --digiSeedOffset2=1664
↪ --digiSteeringConf='StandardSignalOnlyTruth' --AMITag=r9364
↪ --steering="doRDO_TRIG"
↪ --inputHighPtMinbiasHitsFile=/test/digireco/HITS.10701335._0040
↪ 03.pool.root.1,/test/digireco/HITS.10701335._004004.pool.root.1
↪ --inputLowPtMinbiasHitsFile=/test/digireco/HITS.10701323._00861
↪ 0.pool.root.1,/test/digireco/HITS.10701323._008611.pool.root.1,
↪ /test/digireco/HITS.10701323._008612.pool.root.1,/test/digireco
↪ /HITS.10701323._008613.pool.root.1,/test/digireco/HITS.10701323
↪ ._008614.pool.root.1,/test/digireco/HITS.10701323._008615.pool.
↪ root.1,/test/digireco/HITS.10701323._008616.pool.root.1
↪ --numberOfCavernBkg=0 --numberOfHighPtMinBias=0.116075313
↪ --numberOfLowPtMinBias=44.3839246425 --pileupFinalBunch=6
↪ --outputRDO_TRIGFile=RDOTrigger.12118684._001664.pool.root.1
↪ --jobNumber=1664 --triggerConfig="RDOtoRDOTrigger=MCRECO:DBF:TR
↪ IGGERDBMC:2136,35,160" &> output.txt
↪ &
```

A.6.11 Digitisation and reconstruction 2

Athena version: asetup –cmtconfig=x86_64-slc6-gcc62-opt AtlasOffline,21.0.20

The command line input was:

```
Reco_tf.py
↪ --inputHITSFile=/test/digireco_new/HITS.12425264._000086.pool.r
↪ oot.1,/test/digireco_new/HITS.12425264._000087.pool.root.1
↪ --maxEvents=2000 --postExec
↪ 'all:CfgMgr.MessageSvc().setError+=["HepMcParticleLink"]'
↪ 'ESDtoAOD:fixedAttrib=[s if "CONTAINER_SPLITLEVEL =
↪ '"'"'99'"'"'" not in s else "" for s in
↪ svcMgr.AthenaPoolCnvSvc.PoolAttributes];
svcMgr.AthenaPoolCnvSvc.PoolAttributes=fixedAttrib'
↪ 'RDOtoRDOTrigger:conddb.addOverride("/CALO/Ofl/Noise/PileUpNois
↪ eLumi","CALOOflNoisePileUpNoiseLumi-mc15-mu30-dt25ns")'
↪ 'ESDtoAOD:CILMergeAOD.removeItem("xAOD::CaloClusterAuxContainer
↪ #CaloCalTopoClustersAux.LATERAL.LONGITUDINAL.SECOND_R.SECOND_LA
↪ MBDA.CENTER_MAG.CENTER_LAMBDA.FIRST_ENG_DENS.ENG_FRAC_MAX.ISOLA
↪ TION.ENG_BAD_CELLS.N_BAD_CELLS.BADLARQ_FRAC.ENG_BAD_HV_CELLS.N_
↪ BAD_HV_CELLS.ENG_POS.SIGNIFICANCE.CELL_SIGNIFICANCE.CELL_SIG_SA
↪ MPLING.AVG_LAR_Q.AVG_TILE_Q.EM_PROBABILITY.PTD.BadChannelList");
CILMergeAOD.add("xAOD::CaloClusterAuxContainer#CaloCalTopoClustersA
↪ ux.N_BAD_CELLS.ENG_BAD_CELLS.BADLARQ_FRAC.AVG_TILE_Q.AVG_LAR_Q.
↪ CENTER_MAG.ENG_POS.CENTER_LAMBDA.SECOND_LAMBDA.SECOND_R.ISOLATI
↪ ON.EM_PROBABILITY");
StreamAOD.ItemList=CILMergeAOD()' --postInclude
↪ default:PyJobTransforms/UseFrontier.py --preExec
↪ 'all:rec.Commissioning.set_Value_and_Lock(True);
from AthenaCommon.BeamFlags import jobproperties;
jobproperties.Beam.numberOfCollisions.set_Value_and_Lock(20.0);
from LArROD.LArRODFlags import larRODFlags;
larRODFlags.NumberOfCollisions.set_Value_and_Lock(20);
larRODFlags.nSamples.set_Value_and_Lock(4);
larRODFlags.doOFCPileupOptimization.set_Value_and_Lock(True);
larRODFlags.firstSample.set_Value_and_Lock(0);
larRODFlags.useHighestGainAutoCorr.set_Value_and_Lock(True)'
↪ 'all:from TriggerJobOpts.TriggerFlags import TriggerFlags as TF;
TF.run2Config='"'"'2016'"'"'' 'RAWtoESD:from
↪ InDetRecExample.InDetJobProperties import InDetFlags;
 InDetFlags.cutLevel.set_Value_and_Lock(14);
 from JetRec import JetRecUtils;
```

```
f=lambda s:["xAOD::JetContainer#AntiKt4\%sJets"\%(s,),"xAOD::JetAux
 ↪  Container#AntiKt4\%sJetsAux."\%(s,),"xAOD::EventShape#Kt4\%sEve
 ↪  ntShape"\%(s,),"xAOD::EventShapeAuxInfo#Kt4\%sEventShapeAux."\%
 ↪  (s,),"xAOD::EventShape#Kt4\%sOriginEventShape"\%(s,),"xAOD::Eve
 ↪  ntShapeAuxInfo#Kt4\%sOriginEventShapeAux."\%(s,)];
 JetRecUtils.retrieveAODList = lambda :
  ↪  f("EMPFlow")+f("LCTopo")+f("EMTopo")+["xAOD::EventShape#Neutra
  ↪  lParticleFlowIsoCentralEventShape","xAOD::EventShapeAuxInfo#Ne
  ↪  utralParticleFlowIsoCentralEventShapeAux.",
  ↪  "xAOD::EventShape#NeutralParticleFlowIsoForwardEventShape","xA
  ↪  OD::EventShapeAuxInfo#NeutralParticleFlowIsoForwardEventShapeA
  ↪  ux.",
  ↪  "xAOD::EventShape#ParticleFlowIsoCentralEventShape","xAOD::Eve
  ↪  ntShapeAuxInfo#ParticleFlowIsoCentralEventShapeAux.",
  ↪  "xAOD::EventShape#ParticleFlowIsoForwardEventShape","xAOD::Eve
  ↪  ntShapeAuxInfo#ParticleFlowIsoForwardEventShapeAux.",
  ↪  "xAOD::EventShape#TopoClusterIsoCentralEventShape","xAOD::Even
  ↪  tShapeAuxInfo#TopoClusterIsoCentralEventShapeAux.",
  ↪  "xAOD::EventShape#TopoClusterIsoForwardEventShape","xAOD::Even
  ↪  tShapeAuxInfo#TopoClusterIsoForwardEventShapeAux.","xAOD::Calo
  ↪  ClusterContainer#EMOriginTopoClusters","xAOD::ShallowAuxContai
  ↪  ner#EMOriginTopoClustersAux.","xAOD::CaloClusterContainer#LCOr
  ↪  iginTopoClusters","xAOD::ShallowAuxContainer#LCOriginTopoClust
  ↪  ersAux."];
 from eflowRec.eflowRecFlags import jobproperties;
 jobproperties.eflowRecFlags.useAODReductionClusterMomentList.set_V
  ↪  alue_and_Lock(True);
 from TriggerJobOpts.TriggerFlags import TriggerFlags;
TriggerFlags.AODEDMSet.set_Value_and_Lock("AODSLIM");
' 'all:from BTagging.BTaggingFlags import BTaggingFlags;
BTaggingFlags.btaggingAODList=["xAOD::BTaggingContainer#BTagging_An
 ↪  tiKt4EMTopo","xAOD::BTaggingAuxContainer#BTagging_AntiKt4EMTopo
 ↪  Aux.","xAOD::BTagVertexContainer#BTagging_AntiKt4EMTopoJFVtx","
 ↪  xAOD::BTagVertexAuxContainer#BTagging_AntiKt4EMTopoJFVtxAux.","
 ↪  xAOD::VertexContainer#BTagging_AntiKt4EMTopoSecVtx","xAOD::Vert
 ↪  exAuxContainer#BTagging_AntiKt4EMTopoSecVtxAux.-vxTrackAtVertex
 ↪  "];
' 'ESDtoAOD:from ParticleBuilderOptions.AODFlags import AODFlags;
 AODFlags.ThinGeantTruth.set_Value_and_Lock(True);
  AODFlags.ThinNegativeEnergyCaloClusters.set_Value_and_Lock(True);
 AODFlags.ThinNegativeEnergyNeutralPFOs.set_Value_and_Lock(True);
 from JetRec import JetRecUtils;
 aodlist = JetRecUtils.retrieveAODList();
```

```
 JetRecUtils.retrieveAODList = lambda : [item for item in aodlist
  ↪  if not "OriginTopoClusters" in item];
' --preInclude HITtoRDO:Digitization/ForceUseOfPileUpTools.py,Simul
↪  ationJobOptions/preInclude.PileUpBunchTrainsMC15_2015_25ns_Conf
↪  ig1.py,RunDependentSimData/configLumi_run284500_mc16a.py
↪  --skipEvents=0 --autoConfiguration=everything --conditionsTag
↪  default:OFLCOND-MC16-SDR-16
↪  --geometryVersion=default:ATLAS-R2-2016-01-00-01
↪  --runNumber=300307 --digiSeedOffset1=23 --digiSeedOffset2=23
↪  --digiSteeringConf=StandardSignalOnlyTruth --AMITag=r9364
↪  --steering=doRDO_TRIG --inputHighPtMinbiasHitsFile=/test/digire
↪  co_new/HITS.10701335._000080.pool.root.1,/test/digireco_new/HIT
↪  S.10701335._000081.pool.root.1
↪  --inputLowPtMinbiasHitsFile=/test/digireco_new/HITS.10701323._0
↪  01011.pool.root.1,/test/digireco_new/HITS.10701323._001012.pool
↪  .root.1,/test/digireco_new/HITS.10701323._001013.pool.root.1,/t
↪  est/digireco_new/HITS.10701323._001014.pool.root.1,/test/digire
↪  co_new/HITS.10701323._001015.pool.root.1,/test/digireco_new/HIT
↪  S.10701323._001016.pool.root.1 --numberOfCavernBkg=0
↪  --numberOfHighPtMinBias=0.116075313
↪  --numberOfLowPtMinBias=44.3839246425 --pileupFinalBunch=6
↪  --outputAODFile=AOD.12425266._000023.pool.root.1 --jobNumber=23
↪  --triggerConfig=RDOtoRDOTrigger=MCRECO:DBF:TRIGGERDBMC:2136,35,
↪  160  &> output.txt
↪  &
```

A.6.12 Digitisation and reconstruction 3

Fetching the remote input data from EOS. Only single core. Athena version: asetup –cmtconfig=x86_64-slc6-gcc62-opt AtlasOffline,21.0.20

The command line input was:

```
export ATHENA_PROC_NUMBER=0
Reco_tf.py --inputHITSFile=root://eosatlas.cern.ch:1094//eos/atlas/
↪  atlasscratchdisk/rucio/mc16_13TeV/37/41/HITS.12425264._000086.p
↪  ool.root.1,root://eosatlas.cern.ch:1094//eos/atlas/atlasscratch
↪  disk/rucio/mc16_13TeV/a3/85/HITS.12425264._000087.pool.root.1
↪  --maxEvents=200 --postExec
↪  'all:CfgMgr.MessageSvc().setError+=["HepMcParticleLink"]'
↪  'ESDtoAOD:fixedAttrib=[s if "CONTAINER_SPLITLEVEL =
↪  '"'"'99'"'"'" not in s else "" for s in
↪  svcMgr.AthenaPoolCnvSvc.PoolAttributes];
```

```
svcMgr.AthenaPoolCnvSvc.PoolAttributes=fixedAttrib'
 ↪  'RDOtoRDOTrigger:conddb.addOverride("/CALO/Ofl/Noise/PileUpNois
 ↪  eLumi","CALOOflNoisePileUpNoiseLumi-mc15-mu30-dt25ns")'
 ↪  'ESDtoAOD:CILMergeAOD.removeItem("xAOD::CaloClusterAuxContainer
 ↪  #CaloCalTopoClustersAux.LATERAL.LONGITUDINAL.SECOND_R.SECOND_LA
 ↪  MBDA.CENTER_MAG.CENTER_LAMBDA.FIRST_ENG_DENS.ENG_FRAC_MAX.ISOLA
 ↪  TION.ENG_BAD_CELLS.N_BAD_CELLS.BADLARQ_FRAC.ENG_BAD_HV_CELLS.N_
 ↪  BAD_HV_CELLS.ENG_POS.SIGNIFICANCE.CELL_SIGNIFICANCE.CELL_SIG_SA
 ↪  MPLING.AVG_LAR_Q.AVG_TILE_Q.EM_PROBABILITY.PTD.BadChannelList");
CILMergeAOD.add("xAOD::CaloClusterAuxContainer#CaloCalTopoClustersA
 ↪  ux.N_BAD_CELLS.ENG_BAD_CELLS.BADLARQ_FRAC.AVG_TILE_Q.AVG_LAR_Q.
 ↪  CENTER_MAG.ENG_POS.CENTER_LAMBDA.SECOND_LAMBDA.SECOND_R.ISOLATI
 ↪  ON.EM_PROBABILITY");
StreamAOD.ItemList=CILMergeAOD()' --postInclude
 ↪  default:PyJobTransforms/UseFrontier.py --preExec
 ↪  'all:rec.Commissioning.set_Value_and_Lock(True);
from AthenaCommon.BeamFlags import jobproperties;
jobproperties.Beam.numberOfCollisions.set_Value_and_Lock(20.0);
from LArROD.LArRODFlags import larRODFlags;
larRODFlags.NumberOfCollisions.set_Value_and_Lock(20);
larRODFlags.nSamples.set_Value_and_Lock(4);
larRODFlags.doOFCPileupOptimization.set_Value_and_Lock(True);
larRODFlags.firstSample.set_Value_and_Lock(0);
larRODFlags.useHighestGainAutoCorr.set_Value_and_Lock(True)'
 ↪  'all:from TriggerJobOpts.TriggerFlags import TriggerFlags as TF;
TF.run2Config='"'"'2016'"'"' 'RAWtoESD:from
 ↪  InDetRecExample.InDetJobProperties import InDetFlags;
 InDetFlags.cutLevel.set_Value_and_Lock(14);
 from JetRec import JetRecUtils;
f=lambda s:["xAOD::JetContainer#AntiKt4\%sJets"\%(s,),"xAOD::JetAux
 ↪  Container#AntiKt4\%sJetsAux."\%(s,),"xAOD::EventShape#Kt4\%sEve
 ↪  ntShape"\%(s,),"xAOD::EventShapeAuxInfo#Kt4\%sEventShapeAux."\%
 ↪  (s,),"xAOD::EventShape#Kt4\%sOriginEventShape"\%(s,),"xAOD::Eve
 ↪  ntShapeAuxInfo#Kt4\%sOriginEventShapeAux."\%(s,)];
```

```
 JetRecUtils.retrieveAODList = lambda :
 ↪  f("EMPFlow")+f("LCTopo")+f("EMTopo")+["xAOD::EventShape#Neutra
 ↪  lParticleFlowIsoCentralEventShape","xAOD::EventShapeAuxInfo#Ne
 ↪  utralParticleFlowIsoCentralEventShapeAux.",
 ↪  "xAOD::EventShape#NeutralParticleFlowIsoForwardEventShape","xA
 ↪  OD::EventShapeAuxInfo#NeutralParticleFlowIsoForwardEventShapeA
 ↪  ux.",
 ↪  "xAOD::EventShape#ParticleFlowIsoCentralEventShape","xAOD::Eve
 ↪  ntShapeAuxInfo#ParticleFlowIsoCentralEventShapeAux.",
 ↪  "xAOD::EventShape#ParticleFlowIsoForwardEventShape","xAOD::Eve
 ↪  ntShapeAuxInfo#ParticleFlowIsoForwardEventShapeAux.",
 ↪  "xAOD::EventShape#TopoClusterIsoCentralEventShape","xAOD::Even
 ↪  tShapeAuxInfo#TopoClusterIsoCentralEventShapeAux.",
 ↪  "xAOD::EventShape#TopoClusterIsoForwardEventShape","xAOD::Even
 ↪  tShapeAuxInfo#TopoClusterIsoForwardEventShapeAux.","xAOD::Calo
 ↪  ClusterContainer#EMOriginTopoClusters","xAOD::ShallowAuxContai
 ↪  ner#EMOriginTopoClustersAux.","xAOD::CaloClusterContainer#LCOr
 ↪  iginTopoClusters","xAOD::ShallowAuxContainer#LCOriginTopoClust
 ↪  ersAux."];
 from eflowRec.eflowRecFlags import jobproperties;
 jobproperties.eflowRecFlags.useAODReductionClusterMomentList.set_V
 ↪  alue_and_Lock(True);
 from TriggerJobOpts.TriggerFlags import TriggerFlags;
TriggerFlags.AODEDMSet.set_Value_and_Lock("AODSLIM");
' 'all:from BTagging.BTaggingFlags import BTaggingFlags;
BTaggingFlags.btaggingAODList=["xAOD::BTaggingContainer#BTagging_An
↪  tiKt4EMTopo","xAOD::BTaggingAuxContainer#BTagging_AntiKt4EMTopo
↪  Aux.","xAOD::BTagVertexContainer#BTagging_AntiKt4EMTopoJFVtx","
↪  xAOD::BTagVertexAuxContainer#BTagging_AntiKt4EMTopoJFVtxAux.","
↪  xAOD::VertexContainer#BTagging_AntiKt4EMTopoSecVtx","xAOD::Vert
↪  exAuxContainer#BTagging_AntiKt4EMTopoSecVtxAux.-vxTrackAtVertex
↪  "];
' 'ESDtoAOD:from ParticleBuilderOptions.AODFlags import AODFlags;
 AODFlags.ThinGeantTruth.set_Value_and_Lock(True);
  AODFlags.ThinNegativeEnergyCaloClusters.set_Value_and_Lock(True);
 AODFlags.ThinNegativeEnergyNeutralPFOs.set_Value_and_Lock(True);
 from JetRec import JetRecUtils;
 aodlist = JetRecUtils.retrieveAODList();
 JetRecUtils.retrieveAODList = lambda : [item for item in aodlist
 ↪  if not "OriginTopoClusters" in item];
```

```
' --preInclude HITtoRDO:Digitization/ForceUseOfPileUpTools.py,Simul
↪  ationJobOptions/preInclude.PileUpBunchTrainsMC15_2015_25ns_Conf
↪  ig1.py,RunDependentSimData/configLumi_run284500_mc16a.py
↪  --skipEvents=0 --autoConfiguration=everything --conditionsTag
↪  default:OFLCOND-MC16-SDR-16
↪  --geometryVersion=default:ATLAS-R2-2016-01-00-01
↪  --runNumber=300307 --digiSeedOffset1=23 --digiSeedOffset2=23
↪  --digiSteeringConf=StandardSignalOnlyTruth --AMITag=r9364
↪  --steering=doRDO_TRIG --inputHighPtMinbiasHitsFile=root://eosat
↪  las.cern.ch:1094//eos/atlas/atlasscratchdisk/rucio/mc16_13TeV/0
↪  9/44/HITS.10701335._000080.pool.root.1,root://eosatlas.cern.ch:
↪  1094//eos/atlas/atlasscratchdisk/rucio/mc16_13TeV/1a/79/HITS.10
↪  701335._000081.pool.root.1
↪  --inputLowPtMinbiasHitsFile=root://eosatlas.cern.ch:1094//eos/a
↪  tlas/atlasscratchdisk/rucio/mc16_13TeV/6b/c7/HITS.10701323._001
↪  011.pool.root.1,root://eosatlas.cern.ch:1094//eos/atlas/atlassc
↪  ratchdisk/rucio/mc16_13TeV/cf/e6/HITS.10701323._001012.pool.roo
↪  t.1,root://eosatlas.cern.ch:1094//eos/atlas/atlasscratchdisk/ru
↪  cio/mc16_13TeV/a7/01/HITS.10701323._001013.pool.root.1,root://e
↪  osatlas.cern.ch:1094//eos/atlas/atlasscratchdisk/rucio/mc16_13T
↪  eV/5e/1c/HITS.10701323._001014.pool.root.1,root://eosatlas.cern
↪  .ch:1094//eos/atlas/atlasscratchdisk/rucio/mc16_13TeV/ac/b4/HIT
↪  S.10701323._001015.pool.root.1,root://eosatlas.cern.ch:1094//eo
↪  s/atlas/atlasscratchdisk/rucio/mc16_13TeV/7d/35/HITS.10701323._
↪  001016.pool.root.1 --numberOfCavernBkg=0
↪  --numberOfHighPtMinBias=0.116075313
↪  --numberOfLowPtMinBias=44.3839246425 --pileupFinalBunch=6
↪  --outputAODFile=AOD.12425266._000023.pool.root.1 --jobNumber=23
↪  --triggerConfig=RDOtoRDOTrigger=MCRECO:DBF:TRIGGERDBMC:2136,35,
↪  160  &> output.txt
↪  &
```

A.6.13 Digitisation and reconstruction 4

Fetching the remote input data from T-Systems. Only single core. Athena version: asetup –cmtconfig=x86_64-slc6-gcc62-opt AtlasOffline,21.0.20

The command line input was:

```
export ATHENA_PROC_NUMBER=0
```

```
Reco_tf.py --inputHITSFile=https://obs.eu-de.otc.t-systems.com/tsy-
↪ bucket1/HITS.12425264._000086.pool.root.1,https://obs.eu-de.otc
↪ .t-systems.com/tsy-bucket1/HITS.12425264._000087.pool.root.1
↪ --maxEvents=200 --postExec
↪ 'all:CfgMgr.MessageSvc().setError+=["HepMcParticleLink"]'
↪ 'ESDtoAOD:fixedAttrib=[s if "CONTAINER_SPLITLEVEL =
↪ '"'"'99'"'"'" not in s else "" for s in
↪ svcMgr.AthenaPoolCnvSvc.PoolAttributes];
svcMgr.AthenaPoolCnvSvc.PoolAttributes=fixedAttrib'
↪ 'RDOtoRDOTrigger:conddb.addOverride("/CALO/Ofl/Noise/PileUpNois
↪ eLumi","CALOOflNoisePileUpNoiseLumi-mc15-mu30-dt25ns")'
↪ 'ESDtoAOD:CILMergeAOD.removeItem("xAOD::CaloClusterAuxContainer
↪ #CaloCalTopoClustersAux.LATERAL.LONGITUDINAL.SECOND_R.SECOND_LA
↪ MBDA.CENTER_MAG.CENTER_LAMBDA.FIRST_ENG_DENS.ENG_FRAC_MAX.ISOLA
↪ TION.ENG_BAD_CELLS.N_BAD_CELLS.BADLARQ_FRAC.ENG_BAD_HV_CELLS.N_
↪ BAD_HV_CELLS.ENG_POS.SIGNIFICANCE.CELL_SIGNIFICANCE.CELL_SIG_SA
↪ MPLING.AVG_LAR_Q.AVG_TILE_Q.EM_PROBABILITY.PTD.BadChannelList");
CILMergeAOD.add("xAOD::CaloClusterAuxContainer#CaloCalTopoClustersA
↪ ux.N_BAD_CELLS.ENG_BAD_CELLS.BADLARQ_FRAC.AVG_TILE_Q.AVG_LAR_Q.
↪ CENTER_MAG.ENG_POS.CENTER_LAMBDA.SECOND_LAMBDA.SECOND_R.ISOLATI
↪ ON.EM_PROBABILITY");
StreamAOD.ItemList=CILMergeAOD()' --postInclude
↪ default:PyJobTransforms/UseFrontier.py --preExec
↪ 'all:rec.Commissioning.set_Value_and_Lock(True);
from AthenaCommon.BeamFlags import jobproperties;
jobproperties.Beam.numberOfCollisions.set_Value_and_Lock(20.0);
from LArROD.LArRODFlags import larRODFlags;
larRODFlags.NumberOfCollisions.set_Value_and_Lock(20);
larRODFlags.nSamples.set_Value_and_Lock(4);
larRODFlags.doOFCPileupOptimization.set_Value_and_Lock(True);
larRODFlags.firstSample.set_Value_and_Lock(0);
larRODFlags.useHighestGainAutoCorr.set_Value_and_Lock(True)'
↪ 'all:from TriggerJobOpts.TriggerFlags import TriggerFlags as TF;
TF.run2Config='"'"'2016'"'"'' 'RAWtoESD:from
↪ InDetRecExample.InDetJobProperties import InDetFlags;
 InDetFlags.cutLevel.set_Value_and_Lock(14);
 from JetRec import JetRecUtils;
f=lambda s:["xAOD::JetContainer#AntiKt4%sJets"%(s,),"xAOD::JetAuxCo
↪ ntainer#AntiKt4%sJetsAux."%(s,),"xAOD::EventShape#Kt4%sEventSha
↪ pe"%(s,),"xAOD::EventShapeAuxInfo#Kt4%sEventShapeAux."%(s,),"xA
↪ OD::EventShape#Kt4%sOriginEventShape"%(s,),"xAOD::EventShapeAux
↪ Info#Kt4%sOriginEventShapeAux."%(s,)];
```

```
 JetRecUtils.retrieveAODList = lambda :
 ↪  f("EMPFlow")+f("LCTopo")+f("EMTopo")+["xAOD::EventShape#Neutra
 ↪  lParticleFlowIsoCentralEventShape","xAOD::EventShapeAuxInfo#Ne
 ↪  utralParticleFlowIsoCentralEventShapeAux.",
 ↪  "xAOD::EventShape#NeutralParticleFlowIsoForwardEventShape","xA
 ↪  OD::EventShapeAuxInfo#NeutralParticleFlowIsoForwardEventShapeA
 ↪  ux.",
 ↪  "xAOD::EventShape#ParticleFlowIsoCentralEventShape","xAOD::Eve
 ↪  ntShapeAuxInfo#ParticleFlowIsoCentralEventShapeAux.",
 ↪  "xAOD::EventShape#ParticleFlowIsoForwardEventShape","xAOD::Eve
 ↪  ntShapeAuxInfo#ParticleFlowIsoForwardEventShapeAux.",
 ↪  "xAOD::EventShape#TopoClusterIsoCentralEventShape","xAOD::Even
 ↪  tShapeAuxInfo#TopoClusterIsoCentralEventShapeAux.",
 ↪  "xAOD::EventShape#TopoClusterIsoForwardEventShape","xAOD::Even
 ↪  tShapeAuxInfo#TopoClusterIsoForwardEventShapeAux.","xAOD::Calo
 ↪  ClusterContainer#EMOriginTopoClusters","xAOD::ShallowAuxContai
 ↪  ner#EMOriginTopoClustersAux.","xAOD::CaloClusterContainer#LCOr
 ↪  iginTopoClusters","xAOD::ShallowAuxContainer#LCOriginTopoClust
 ↪  ersAux."];
 from eflowRec.eflowRecFlags import jobproperties;
 jobproperties.eflowRecFlags.useAODReductionClusterMomentList.set_V
 ↪  alue_and_Lock(True);
 from TriggerJobOpts.TriggerFlags import TriggerFlags;
TriggerFlags.AODEDMSet.set_Value_and_Lock("AODSLIM");
' 'all:from BTagging.BTaggingFlags import BTaggingFlags;
BTaggingFlags.btaggingAODList=["xAOD::BTaggingContainer#BTagging_An
 ↪  tiKt4EMTopo","xAOD::BTaggingAuxContainer#BTagging_AntiKt4EMTopo
 ↪  Aux.","xAOD::BTagVertexContainer#BTagging_AntiKt4EMTopoJFVtx","
 ↪  xAOD::BTagVertexAuxContainer#BTagging_AntiKt4EMTopoJFVtxAux.","
 ↪  xAOD::VertexContainer#BTagging_AntiKt4EMTopoSecVtx","xAOD::Vert
 ↪  exAuxContainer#BTagging_AntiKt4EMTopoSecVtxAux.-vxTrackAtVertex
 ↪  "];
' 'ESDtoAOD:from ParticleBuilderOptions.AODFlags import AODFlags;
 AODFlags.ThinGeantTruth.set_Value_and_Lock(True);
  AODFlags.ThinNegativeEnergyCaloClusters.set_Value_and_Lock(True);
 AODFlags.ThinNegativeEnergyNeutralPFOs.set_Value_and_Lock(True);
 from JetRec import JetRecUtils;
 aodlist = JetRecUtils.retrieveAODList();
 JetRecUtils.retrieveAODList = lambda : [item for item in aodlist
 ↪  if not "OriginTopoClusters" in item];
```

```
' --preInclude HITtoRDO:Digitization/ForceUseOfPileUpTools.py,Simul
↪  ationJobOptions/preInclude.PileUpBunchTrainsMC15_2015_25ns_Conf
↪  ig1.py,RunDependentSimData/configLumi_run284500_mc16a.py
↪  --skipEvents=0 --autoConfiguration=everything --conditionsTag
↪  default:OFLCOND-MC16-SDR-16
↪  --geometryVersion=default:ATLAS-R2-2016-01-00-01
↪  --runNumber=300307 --digiSeedOffset1=23 --digiSeedOffset2=23
↪  --digiSteeringConf=StandardSignalOnlyTruth --AMITag=r9364
↪  --steering=doRDO_TRIG --inputHighPtMinbiasHitsFile=https://obs.
↪  eu-de.otc.t-systems.com/tsy-bucket1/HITS.10701335._000080.pool.
↪  root.1,https://obs.eu-de.otc.t-systems.com/tsy-bucket1/HITS.107
↪  01335._000081.pool.root.1
↪  --inputLowPtMinbiasHitsFile=https://obs.eu-de.otc.t-systems.com
↪  /tsy-bucket1/HITS.10701323._001011.pool.root.1,https://obs.eu-d
↪  e.otc.t-systems.com/tsy-bucket1/HITS.10701323._001012.pool.root
↪  .1,https://obs.eu-de.otc.t-systems.com/tsy-bucket1/HITS.1070132
↪  3._001013.pool.root.1,https://obs.eu-de.otc.t-systems.com/tsy-b
↪  ucket1/HITS.10701323._001014.pool.root.1,https://obs.eu-de.otc.
↪  t-systems.com/tsy-bucket1/HITS.10701323._001015.pool.root.1,htt
↪  ps://obs.eu-de.otc.t-systems.com/tsy-bucket1/HITS.10701323._001
↪  016.pool.root.1 --numberOfCavernBkg=0
↪  --numberOfHighPtMinBias=0.116075313
↪  --numberOfLowPtMinBias=44.3839246425 --pileupFinalBunch=6
↪  --outputAODFile=AOD.12425266._000023.pool.root.1 --jobNumber=23
↪  --triggerConfig=RDOtoRDOTrigger=MCRECO:DBF:TRIGGERDBMC:2136,35,
↪  160  &> output.txt
↪  &
```

A.6.14 Digitisation and reconstruction 5

Repeat of Subsection A.6.13.

A.7 Hardware

More details on the hardware specifications of the utilised VMs is given below.

A.7.1 Göttingen

One VM from Göttingen was used.

PCATLAS46

The VM was on an hypervisor, to which no other user had access. It was a completely controlled environment. Therefore, the tests that were performed on PCATLAS46 could

not have been influenced by neighbouring VMs.

Listing 6: "lscpu" result on PCATLAS46 at Göttingen.

```
1  [root@pcatlas46 ~]# lscpu
2  Architecture:          x86_64
3  CPU op-mode(s):        32-bit, 64-bit
4  Byte Order:            Little Endian
5  CPU(s):                8
6  On-line CPU(s) list:   0-7
7  Thread(s) per core:    1
8  Core(s) per socket:    4
9  Socket(s):             2
10 NUMA node(s):          1
11 Vendor ID:             GenuineIntel
12 CPU family:            6
13 Model:                 15
14 Model name:            Intel(R) Xeon(R) CPU           X5355  @
     2.66GHz
15 Stepping:              11
16 CPU MHz:               1999.998
17 BogoMIPS:              5333.37
18 Virtualization:        VT-x
19 L1d cache:             32K
20 L1i cache:             32K
21 L2 cache:              4096K
22 NUMA node0 CPU(s):     0-7
```

A.7.2 CERN

The details of the CERN VMs follow in this subsection.

DataCloud14

The VM was on an hypervisor, to which no other user had access. It was a completely controlled environment. Therefore, the tests that were performed on DataCloud14 could not have been influenced by neighbouring VMs.

The hypervisor was within the CERN OpenStack cluster.

Clouddata02

This 4-core VM was on a shared hypervisor within the OpenStack cluster. Some initial profiling was performed on this VM, before the dedicated infrastructures became available. The profiling results are therefore only accurate to a certain degree, but have

been repeated multiple times and verified on the dedicated machines, in order to exclude interference from neighbouring VMs.

Listing 7: "lscpu" result on Clouddata02 at CERN.

```
1  [root@datacloud02 ~]# lscpu
2  Architecture:          x86_64
3  CPU op-mode(s):        32-bit, 64-bit
4  Byte Order:            Little Endian
5  CPU(s):                4
6  On-line CPU(s) list:   0-3
7  Thread(s) per core:    1
8  Core(s) per socket:    1
9  Socket(s):             4
10 NUMA node(s):          1
11 Vendor ID:             GenuineIntel
12 CPU family:            6
13 Model:                 44
14 Model name:            Intel(R) Xeon(R) CPU           L5640  @
     2.27GHz
15 Stepping:              2
16 CPU MHz:               2266.746
17 BogoMIPS:              4533.49
18 Hypervisor vendor:     KVM
19 Virtualization type:   full
20 L1d cache:             32K
21 L1i cache:             32K
22 L2 cache:              256K
23 L3 cache:              12288K
24 NUMA node0 CPU(s):     0-3
```

A.7.3 Exoscale

The Exoscale VMs were of the type 'Huge', with eight CPU cores and 32 GB of RAM. In order to have a better performance comparison with the other providers, the available RAM was limited to 16 GB before each test. How it was done, can be found in Listing 1 in the appendix.

Listing 8: "lscpu" result on an Exoscale VM.

```
1  Architecture:          x86_64
2  CPU op-mode(s):        32-bit, 64-bit
3  Byte Order:            Little Endian
4  CPU(s):                8
5  On-line CPU(s) list:   0-7
6  Thread(s) per core:    1
```

```
 7 Core(s) per socket:    4
 8 Socket(s):             2
 9 NUMA node(s):          1
10 Vendor ID:             GenuineIntel
11 CPU family:            6
12 Model:                 42
13 Model name:            Intel Xeon E312xx (Sandy Bridge)
14 Stepping:              1
15 CPU MHz:               2399.998
16 BogoMIPS:              4799.99
17 Virtualization:        VT-x
18 Hypervisor vendor:     KVM
19 Virtualization type:   full
20 L1d cache:             32K
21 L1i cache:             32K
22 L2 cache:              4096K
23 NUMA node0 CPU(s):     0-7
24 Flags:                 fpu vme de pse tsc msr pae mce cx8 apic
     sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht
     syscall nx rdtscp lm constant_tsc rep_good nopl eagerfpu pni
     pclmulqdq vmx ssse3 cx16 sse4_1 sse4_2 x2apic popcnt
     tsc_deadline_timer aes xsave avx hypervisor lahf_lm arat
     tpr_shadow vnmi flexpriority ept vpid xsaveopt
```

A.7.4 IBM

The IBM VMs had eight CPU cores and 16 GB of memory.

Listing 9: "lscpu" result on an IBM VM.

```
 1 Architecture:          x86_64
 2 CPU op-mode(s):        32-bit, 64-bit
 3 Byte Order:            Little Endian
 4 CPU(s):                8
 5 On-line CPU(s) list:   0-7
 6 Thread(s) per core:    1
 7 Core(s) per socket:    8
 8 Socket(s):             1
 9 NUMA node(s):          1
10 Vendor ID:             GenuineIntel
11 CPU family:            6
12 Model:                 79
13 Model name:            Intel(R) Xeon(R) CPU E5-2683 v4 @
     2.10GHz
14 Stepping:              1
```

```
15 CPU MHz:              2100.013
16 BogoMIPS:             4200.08
17 Hypervisor vendor:    Xen
18 Virtualization type:  full
19 L1d cache:            32K
20 L1i cache:            32K
21 L2 cache:             256K
22 L3 cache:             40960K
23 NUMA node0 CPU(s):    0-7
```

A.7.5 T-systems

The T-Systems VMs were of the type 'Computing II, c2.2xlarge, 8 vCPUs, 16 GB'. The attached disks were of the size 150 GB.

Listing 10: "lscpu" result a T-Systems VM.

```
1  Architecture:          x86_64
2  CPU op-mode(s):        32-bit, 64-bit
3  Byte Order:            Little Endian
4  CPU(s):                8
5  On-line CPU(s) list:   0-7
6  Thread(s) per core:    1
7  Core(s) per socket:    8
8  Socket(s):             1
9  NUMA node(s):          1
10 Vendor ID:             GenuineIntel
11 CPU family:            6
12 Model:                 63
13 Model name:            Intel(R) Xeon(R) CPU E5-2658A v3 @
     2.20GHz
14 Stepping:              2
15 CPU MHz:               2194.941
16 BogoMIPS:              4389.93
17 Hypervisor vendor:     Xen
18 Virtualization type:   full
19 L1d cache:             32K
20 L1i cache:             32K
21 L2 cache:              256K
22 L3 cache:              30720K
23 NUMA node0 CPU(s):     0-7
24 Flags:                 fpu vme de pse tsc msr pae mce cx8 apic
     sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht
     syscall nx rdtscp lm constant_tsc rep_good nopl eagerfpu pni
     pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe
```

```
popcnt tsc_deadline_timer aes xsave avx f16c rdrand
hypervisor lahf_lm fsgsbase bmi1 avx2 smep bmi2 erms invpcid
xsaveopt
```

A.8 Additional Tables

	Exoscale Wall Time [s]	**IBM Wall Time [s]**	**T-Systems Wall Time [s]**
EvGen	2915 ± 137	3927 ± 118	4089 ± 126
MC Sim	1279 ± 61	2321 ± 353	4808 ± 1298
Reco 1	5737 ± 440	8193 ± 571	±
Reco 2	5700 ± 190	8165 ± 598	15430 ± 2612
Reco 3	4547 ± 478	7061 ± 629	8681 ± 1827
Reco 4	4528 ± 398	7009 ± 604	10785 ± 1534
Reco 5	3147 ± 88	4756 ± 680	7099 ± 1203
Reco 6	3529 ± 451	4919 ± 468	7986 ± 1810
Reco 7	5550 ± 943	7630 ± 760	15348 ± 2886
Digi Reco 1	1210 ± 123	2019 ± 172	2789 ± 524
Digi Reco 2	8381 ± 681	15116 ± 2395	25821 ± 6151

Table A2: Comparing the Clouds.

	Exoscale Wall Time [s]	[%]	**IBM Wall Time [s]**	[%]	**T-Systems Wall Time [s]**	[%]
EvGen	2915 ± 137	4.70	3927 ± 118	3.02	4089 ± 126	3.07
MC Sim	1279 ± 61	4.77	2321 ± 353	15.19	4808 ± 1298	27.00
Reco 1	5737 ± 440	7.68	8193 ± 571	6.97	±	
Reco 2	5700 ± 190	3.34	8165 ± 598	7.32	15430 ± 2612	16.93
Reco 3	4547 ± 478	10.50	7061 ± 629	8.90	8681 ± 1827	21.05
Reco 4	4528 ± 398	8.78	7009 ± 604	8.62	10785 ± 1534	14.22
Reco 5	3147 ± 88	2.79	4756 ± 680	14.31	7099 ± 1203	16.95
Reco 6	3529 ± 451	12.77	4919 ± 468	9.52	7986 ± 1810	22.66
Reco 7	5550 ± 943	16.99	7630 ± 760	9.95	15348 ± 2886	18.80
Digi Reco 1	1210 ± 123	10.21	2019 ± 172	8.52	2789 ± 524	18.77
Digi Reco 2	8381 ± 681	8.12	15116 ± 2395	15.85	25821 ± 6151	23.82

Table A3: Comparing the Clouds. Displayed are the Wall time together with the standard deviation. Behind that, the standard deviation in percent is shown.

Model Error:	EvGen [%]	MC Sim [%]	Reco 1 [%]	Reco 2 [%]	Reco 3 [%]
IBM	0.62	-22.93	-7.40	-1.84	-14.19
TSY	1.88	-60.30	-33.89	-38.00	-28.87
EXO 1	-6.48	-0.41	-3.19	0.88	-5.17
EXO 2	3.09	0.68	-1.44	-0.51	7.32
EXO 3	3.84	5.14	2.88	0.69	

Table A4: Model error on the large scale.

Model difference	Wall Time [%]
EvGen	0.49
MC Sim	2.68
Reconstruction	-0.28

Table A5: Deviation of the model prediction from the measurement, using the VMs at CERN and at Göttingen.

Gerhard Ferdinand Rzehorz ● Pestalozzistr 3 ● 76646 Bruchsal, Deutschland

E-Mail: gerhard.rzehorz@gmx.de ● Telefon: +49 176 2425 5777

Lebenslauf

Persönliche Daten

Name:	Gerhard Ferdinand Rzehorz
Geburtsdatum:	29.01.1987
Geburtsort:	Bruchsal
Staatsangehörigkeit:	Deutsch
Adresse:	Pestalozzistr 3, D-76646 Bruchsal

Akademische Ausbildung

November 2014 - heute	Doktorrand am II. Physikalischen Institut, Georg-August-Universität Göttingen Promotionsthema: Data intensive ATLAS workflows in the Cloud
April 2007 - März 2013	Diplom am Institut für Experimentelle Kern- und Teilchenphysik, Karlsruher Institut für Technologie Thema: Grid-Site-Monitoring at Belle II

Wehrdienst

Juli 2006 - März 2007	Luftwaffenausbildungsbataillon in Germersheim

Schulische Ausbildung

September 1997 - Juni 2006	Justus-Knecht-Gymnasium in Bruchsal Abschluss: Abitur
August 1993 - Juli 1997	Johann-Peter-Hebel Grundschule in Bruchsal Abschluss: Grundschulabschluss